地理信息系统开发专家指南



刘光 唐大仕 编著

Web GIS开发

——ArcGIS Server与.NET



- 国内第一本Web GIS开发实战手册
- 以丰富的实例，系统、全面地介绍了基于Web服务的GIS开发
- 使用C#语言开发基于ArcGIS Server的Web GIS

清华大学出版社



刘 光　唐大仕　编著

Web GIS开发

ArcGIS Server与.NET

清华大学出版社

北　京

内 容 简 介

ArcGIS Server 是功能强大的基于服务器的地理信息系统产品，本书以循序渐进的方式，通过大量的实例介绍如何在 Visual Studio 中，使用 C#语言开发基于 ArcGIS Server 的 Web GIS。全书内容涉及使用 ArcGIS Server 开发 Web GIS 的各个层面，包括 ArcGIS Server 9.2 的功能、架构及安装介绍，ArcGIS Server 的管理、服务的发布以及配置文件的使用，自定义工具与命令的创建，数据源、图形对象类、任务的自定义及操作，ArcGIS 服务器功能的扩展，以及如何直接使用 ArcGIS Server 提供的 Web 服务开发程序并对其进行再封装。最后介绍了 Web GIS 中的安全、部署以及性能调优应考虑的关键问题。

本书适用于政府、企业相关部门的 GIS 研究与开发人员，也适用于高等院校地理学、地理信息系统、房地产、环境科学、资源与城乡规划管理、区域经济学等专业学生参考与学习。本书还适合作为各种 GIS 培训学员的学习教材与参考书。

本书封面贴有清华大学出版社防伪标签，无标签者不得销售。
版权所有，侵权必究。侵权举报电话：010-62782989　13701121933

图书在版编目（CIP）数据

Web GIS开发——ArcGIS Server与.NET/刘光，唐大仕编著. —北京：清华大学出版社，2009
ISBN 978-7-302-19737-9

I. W…　II. ①刘…　②唐…　III. 地理信息系统—应用软件　IV. P208

中国版本图书馆 CIP 数据核字（2009）第 038857 号

责任编辑：夏非彼　卢　亮
装帧设计：图格新知
责任校对：贾淑媛
责任印制：
出版发行：清华大学出版社　　地　址：北京清华大学学研大厦 A 座
http://www.tup.com.cn　　邮　编：100084
社 总 机：010-62770175　　邮　购：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 刷 者：
装 订 者：
经　销：全国新华书店
开　本：190×260　　印　张：25.25　字　数：646 千字
版　次：　　印　次：
印　数：
定　价：

本书如存在文字不清、漏印以及缺页、倒页、脱页等印装质量问题，请与清华大学出版社出版部联系调换。联系电话：（010）62770177 转 3103　产品编号：

前　　言

人类正面临一个海量信息的时代，社会的发展、国家与企业的竞争能力越来越依赖于对信息的占有量与处理和利用能力。而占人类活动全部信息 80%的空间信息如何更快、更好、更充分地发挥作用已成为全球关注的热点。空间信息科学与技术正是在这样一个强烈的需求呼唤与应用积累下产生的一个新兴高技术交叉学科，它结合空间技术、信息技术和各类应用技术，主要研究空间信息的获取、存储、管理、查询、分析、应用、共享、可视表达等理论、方法与技术。“21 世纪是全球数字化的时代”（比尔·盖茨），“十五”期间我国就已将“数字城市”、“数字国土”等列入国民经济和科技发展重点计划，“数字北京”、“数字福建”、“数字黄河”等等一些重大工程即将大展宏图，地理信息系统的研发任重道远。

自 20 世纪 60 年代诞生以来，GIS 发展迅速，应用也日趋深化和广泛，逐步融入信息技术（IT）的主流，正在成为信息产业新的增长点,是发展潜力巨大的地理信息产业的主要组成部分之一。如今 GIS 的应用已经成为我国国民经济和社会信息化建设的亮点，日益深入到各个专业领域和百姓日常生活中。

GIS 经历了单机环境应用向网络环境应用发展的过程，网络环境 GIS 应用从局域网内客户/服务器（Client/Server，C/S）结构的应用向 Internet 环境下浏览器/服务器（Browser/Server，B/S）结构的 Web GIS 应用发展。随着 Internet 的发展，Web GIS 开始逐步成为 GIS 应用的主流，Web GIS 相对于 C/S 结构而言，具有部署方便、使用简单、对网络带宽要求低的特点，为地理信息服务的发展奠定了基础。

然而，早期的 Web GIS 功能较弱，主要用于电子地图的发布和简单的空间分析与数据编辑，难以实现较为复杂的图形交互应用（如 GIS 数据的修改和编辑、制图）和复杂的空间分析，还无法取代传统的 C/S 结构的 GIS 应用，出现了 B/S 结构与 C/S 结构并存的局面，而 C/S 结构涉及客户端与服务器端之间大量数据转输，无法在互联网平台实现复杂的、大规模的地理信息服务。

B/S 结构应用已经由浏览器/网络服务器/数据服务器（Browser/Web Server/Data Server）三层架构阶段进入到浏览器/网络服务器/应用服务器/数据服务器（Browser/Web Server/Application Server/Data Server）四层架构阶段。在新的四层架构中，网络服务器和应用服务器分离，并且其间还可以插入二次开发和扩展功能，其中的应用服务器一般为支持远程调用的组件式 GIS 平台，或由组件式 GIS 平台封装而成。将 GIS 复杂数据分析与处理功能（编辑、拓扑关系的构建、对象关系的自动维护、制图）移到 GIS 应用服务器上，使客户端与服务端的数据传输减少到最少的程度，为在 Internet 上实现复杂、大规模的地理信息服务提供了可能。这一架构带来的巨大优势是使服务器端具有极强的扩展性，因此作为应用服务器的组件式 GIS 所具备的功能，都可以通过 B/S 结构实现，Web GIS 不再是只能满足地图浏览和查询的简单软件了，而是一个体系先进，功能强大的服务器端 GIS（Server GIS）。新的服务器端 GIS 将是未来应用发展的主流。

ArcGIS Server 是一个基于 Web 的企业级 GIS 解决方案，它为创建和管理基于服务器的 GIS

应用提供了一个高效的框架平台。它充分利用了 ArcGIS 的核心组件库 ArcObjects，并且基于工业标准提供 Web GIS 服务。ArcGIS Server 将 GIS 和网络技术（Web）两个先进的技术结合在一起：GIS 擅长与空间相关的分析和处理，网络技术则提供全球互联，促进信息共享。这两项技术协同工作，相得益彰。

本书通过大量的实例，详细介绍了如何利用 ArcGIS Server 开发 Web GIS。

在第 1 章中，介绍了 GIS 及其发展和与 Web 服务的联合，并介绍了基于 Web 服务的 GIS 的相关规范。

第 2 章中详细介绍了 ArcGIS Server 9.2 的主要功能以及产品的分类分级，并介绍了 ArcGIS Server 系统的整体架构及其相关技术，最后介绍了该产品的安装过程。

第 3 章介绍了 ArcGIS Server 的管理、服务的发布以及配置文件的使用。

第 4 章首先介绍了使用 ArcGIS Server 创建 Web GIS 的不同方法，并介绍了开发 Web GIS 的基础，即 Web 应用程序框架，及该框架中 Ajax 的使用，最后通过几个实例介绍了如何创建自定义的工具与命令，实现图形与属性的双向查询与展示。

第 5 章介绍了 ArcGIS Server 中支持的数据源类型及其原理，并着重介绍了 ArcGIS Server 数据源的不同层次的使用与管理。本章还介绍了与数据源对应的资源与功能的操作。

第 6 章通过两个大的实例介绍了如何在 ArcGIS Server 的 Web 应用程序框架（Web ADF）中，无缝使用自定义格式以及存储的数据源。

第 7 章首先介绍了图形对象类及其操作的不同层次，然后分别通过实例介绍如何在 Web 端以及 GIS 服务器端操作图形对象，最后介绍了图形对象在不同层次之间转换的方法。

第 8 章介绍了任务框架的概念、组成，并介绍如何自定义任务框架。

第 9 章介绍如何通过使用 COM 功能对象以及服务器对象扩展两种方式来扩展 ArcGIS 服务器的功能。

第 10 章则介绍了如何不在 ArcGIS Server 的 Web 应用程序框架中，直接使用 ArcGIS Server 提供的 Web 服务开发程序，以及如何对 ArcGIS Server 提供的 Web 服务进行再封装，为其他应用系统提供粗粒度的具有针对性的 Web 方法。

第 11 章介绍了 Web GIS 中应考虑的安全问题及其措施，以及其部署和性能调优方面的问题。

关于本书的实例，读者可以到图格新知网站（www.booksaga.com）下载其对应的源代码和一些相关文件。

参与本书编写的人员有刘光、唐大仕、刘增良、韩光瞬、刘小东、贺小飞、李珍贵、岳江、潘杏花、丁修平、梁宁海、马学坤、胡汝章、潘子南与何军等人。

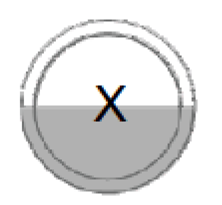
由于作者水平、经验有限，书中不可避免存在一些缺点与错误，希望能够得到广大专家与读者的批评与指正。

作者

2009.3

目　　录

Web GIS



Web GIS

第 1 章



地理信息系统的发展与Web服务

地理信息系统（Geographical Information System，GIS）是一种采集、处理、存储、管理、分析、输出地理空间数据及其属性信息的计算机信息系统。自 20 世纪 60 年代诞生以来，GIS 发展迅速，应用也日趋深化和广泛，逐步融入信息技术（IT）的主流，正在成为信息产业新的增长点，是发展潜力巨大的地理信息产业的主要组成部分之一。如今 GIS 的应用已经成为我国国民经济和社会信息化建设的亮点，日益深入到各个专业领域和百姓日常生活中。

通过本章你将了解到：

1.1 地理信息系统及其发展趋势

随着计算机技术、网络技术、数据库技术等的发展以及应用的不断深化，GIS 技术的发展呈现出新的特点和趋势，基于互联网的 Web GIS 就是其中之一。Web GIS 除了应用于传统的国土、资源、环境等政府管理领域外，也正在促进与老百姓生活息息相关的车载导航、移动位置服务、智能交通、抢险救灾、城市设施管理、现代物流等产业的迅速发展。

1.1.1 地理信息系统的发展

在一定意义上，地理信息系统是计算机和信息系统技术在地理科学中运用发展的产物。因此地理信息系统不仅受地理信息系统的应用和需求的推动，同时也受计算机和信息科学技术的推动。

20 世纪 60 年代末世界上第一个地理信息系统——加拿大地理信息系统（CGIS）诞生，该系统主要用于自然资源的管理和规划；随后，美国哈佛大学研制出 SYMAP 系统。地理信息系统因日益引起各国政府和科学家的高度重视而迅速发展。GIS 的发展经历了 20 世纪 70 年代的大量试验开发阶段，20 世纪 80 年代的商业开发和运作阶段以及 20 世纪 90 年代以用户为主导的阶段。在 GIS 发展初期，只有地理研究人员、地质调查局、土地森林管理部门、人口调查等专业部门和研究人员对其感兴趣，而目前 GIS 已深入到政府管理、城市规划、科学研究、资源开发利用、测绘、军事等广大的领域。21 世纪，地理信息系统已远远不是地理学界或测绘学领域的概念，而将成为人们采集、管理、分析空间数据，共享全球信息资源，为政府管理提供决策，科学研究和实施可持续发展战略的工具和手段。其内涵从狭义的地理信息系统（管理地理信息的计算机系统）到更广泛的空间信息系统（Spatial Information System），并逐渐形成地球信息科学（GeoInformatics）。

从 20 世纪 60 年代以来，计算模式的发展已经经历了单机计算、集中计算到 C/S 模式、B/S 模式（三层结构模式）的不同阶段，正逐渐进入以 Web 服务（Web Services）为主要特征的面向服务的计算模式。

就技术层面而言，地理信息系统的发展也经历了三代，现在正在向第四代过渡。从 GIS 中引入的网络技术方面来看，其中第一代（20 世纪 60 年代—80 年代中期）是以单机单用户为平台、以系统为中心；第二代（20 世纪 80 年代中期—90 年代中期）开始引用网络，实现了多机多用户的 GIS；第三代（20 世纪 80 年代中期—本世纪初）引入了 Internet 技术，开始向以数据为中心的方向过渡，实现了较低层次的（浏览型或简单查询型）的 B/S 结构。

在以前的地理信息系统中，基本上以系统为中心，不同系统之间壁垒比较分明，数据共享与服务共享困难。在三十多年的时间里，形成了许多 GIS 软件，他们在不同的环境中独自发展，有自己的文化背景、领域背景和技术背景，形成了自己的数据模型和功能组织结构。虽然在功能和问题描述、实际操作上差别甚大，加上内部空间数据组织不同或者互相保密，形成了不同的壁垒，为信息共享增加了许多困难。

由于 Internet 技术和 Web 技术的成熟与大规模普及应用，GIS 开始面向传统行业和广大民众，Web GIS 开始出现和发展，并逐渐成为 GIS 应用的一种重要方式。Web GIS 是将 Web 技术应用于

技术应用于 GIS 开发的产物，是一个交互式的、分布式的、动态的地理信息系统，是由多台主机、多个数据库和无数终端，并由客户机与服务器（HTTP 服务器及应用服务器）相连接所组成的。Web GIS 中，空间信息应用主要采取的是 B/S（浏览器/服务器）方式。

1.1.2　传统 Web GIS 的不足

网络技术及分布式计算技术给 GIS 提供了更好的支持，同时也提出了更高的要求。随着网络信息基础设施和技术的不断发展与完善，分布式地理信息服务正成为人们获取地理信息的主要手段。与传统方式相比，分布式地理信息服务具有更广泛的访问范围、平台独立性、低系统成本、更简单的操作等优点，是今后 GIS 发展的重要方向。

但是，传统的 Web GIS 还有相当的不足，主要有如下几点：

（1）Web GIS 的主要功能和应用是用于地图的发布，这类系统基本上是浏览型或功能相对简单的查询型系统。即使有少量的对空间数据的操纵，但这种操纵的功能很弱，无法进行复杂的一体化操作，离全面的互操作及分布式的地理信息系统的要求还很遥远。

（2）Web GIS 中主要是服务端与客户端的通讯，由于服务端与客户端的地位没有形成对等的实体，因而难于建立分布式的地理信息系统。

（3）Web GIS 中传递的数据主要是以矢量形式表达的少量地图数据或者是以栅格形式表达的地图，这样的地图数据，在各个应用系统中的格式不统一，语义也不统一。由于缺乏统一的标准，数据的共享难于实现。

（4）Web GIS 中实现的操作，在各个系统中没有统一的描述的机制（虽然也有一些系统制定了一定的查询语言如 GeoSQL，但这不是所有的系统都采用的），也没有对这些操作和服务提供注册和发现的机制，因此服务的共享难以实现。

（5）Web GIS 还没有形成一套有效的集成机制。新一代的 GIS 要求有效的分布式空间数据管理和计算，包括：多用户同步空间数据操作与处理机制；数据、服务代理和多级 B/S 体系结构；异种 GIS 系统互连与互操作；空间数据分布式存储与数据安全；空间数据高效压缩与解压缩；同时要求强大的应用集成能力，包括有效的遥感、地理信息系统、全球定位系统集成；强大的应用模型支持能力；GIS 与 MIS（管理信息系统）、特别是 ERP（企业资源计划）的有机集成；GIS 与 OA（办公自动化）的有机集成；GIS 与 CAD（计算机辅助设计）的有机集成；GIS 与 DCS（决策支持系统）的有机集成；有一定实时能力、微型化、嵌入式 GIS 与各类设备的集成等等。

从以上所列举的来看，GIS 中大量的数据不断积累、各种层次的软件也越来越多，Web 技术的发展给 GIS 提出了更高的要求，GIS 的分布式、可互操作性显得越来越重要，这恰恰是当前 Web GIS 要着重解决的问题，这也是新一代（即第四代）GIS 的一个重要发展方向。

1.1.3　Web 服务成为解决方案

随着 Web 技术、组件技术、分布式系统等技术的发展，在近几年出现了 Web 服务技术，并逐渐引起了人们的注意，并成为分布式异构 GIS 进行互操作集成的首选技术。

在 Web 应用的不断发展过程中，人们发现在 Web 应用和传统桌面应用（比如企业内部管理系统、办公自动化系统等）之间存在着连接的鸿沟，人们不得不重复地将数据在 Web 应用和传统桌面应用之间迁移，这成为了阻碍 Web 应用进入主流工作流的一个巨大障碍。

从 1998 年开始发展的 XML 技术及其相关技术已证明可以解决这个问题。而近期开始蓬勃发展的 Web 服务技术则正是基于 XML 技术的针对这一问题的最佳解决方案。Web 服务（Web Services）的主要目标就是在现有的各种异构平台的基础上构筑一个通用的与平台无关、语言无关的技术层，各种不同平台之上的应用依靠这个技术层来实施彼此的连接和集成。Web 服务与传统 Web 应用技术的差异在于：传统 Web 应用技术解决的问题是如何让人来使用 Web 应用所提供的服务，而 Web 服务则要解决如何让计算机系统来使用 Web 应用所提供的服务。

将 Web 服务应用于 GIS，则可以使传统的地理信息系统由独立的 C/S 结构或 B/S 结构，实现到基于 Web 服务体系的 GIS 的跨跃（如图 1.1 所示）。

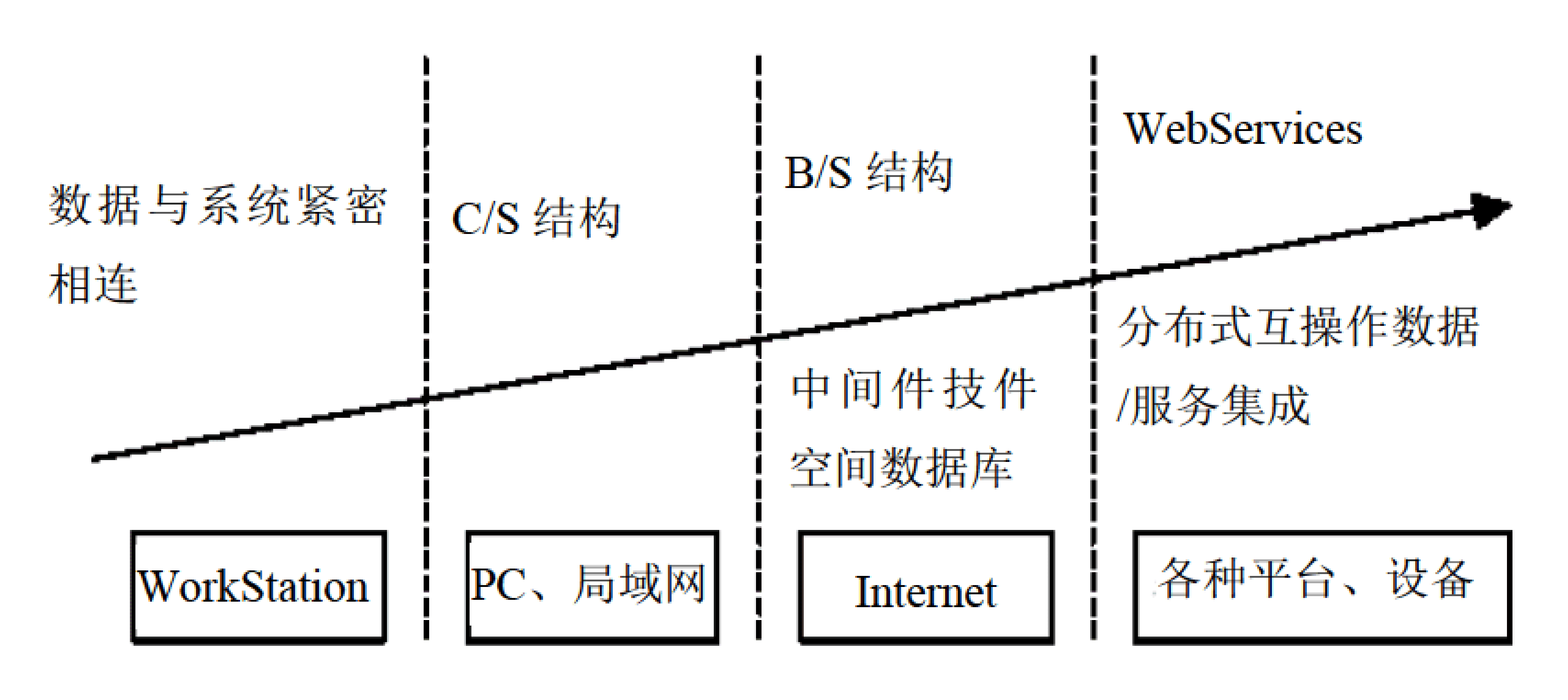


图 1.1　GIS 的网络化的发展趋势

1.2　Web 服务及其特点

1.2.1　Web 服务概述

W3C（Word Wide Web Consortium）组织对 Web 服务的定义如下：

Web 服务是一个由 URI（Uniform Resource Identifer）指定的软件组件或应用，它的接口和绑定可以用标准的 XML 进行描述并支持与其他软件和组件进行交互。

在实现分布式、可互操作及应用系统集成方面，Web 服务技术成为新一代 Web 技术。Web 服务从本质上来说是一种基于对象/组件模型的分布式计算技术。Web 服务的基础是 XML（可扩展标记语言）及基于其上的 SOAP（Simple Object Access Protocol，简单对象访问协议），其基本结构是：客户端和服务端之间把请求和数据结果以 XML 的形式进行 SOAP 包装，以 HTTP 等形式进行传送，从而实现相应交互。Web 服务技术的一大特点是，通过使用 Web 服务定义接口，可以掩盖各种不同实现之间的区别以及各相互联结的系统之间的异构性。在 Web 服务技术中，整个网

络成为一个开放式的组件平台，通过组合不同的 Web 组件，应用程序很容易就能得到近乎无限的扩展，从而满足用户的各种功能需求。

也就是说，Web 服务就是由服务组件通过某些网络协议提供的远程调用接口。Web Services 并不是一种新的服务端组件，而是原来的服务端组件提供了一种新的通过 SOAP 协议来调用的统一接口。

1.2.2　Web 服务的特点

Web 服务重要之处在于，Web 服务实现平台的细节和业务调用程序无关。Web Services 可以用其声明的 API 和调用机制（网络、数据编码模式等）进行访问。这种方式类似于浏览器和 Web 应用服务器之间的关系，Web 服务器不必关心使用它是哪类客户——可以是不同种类的浏览器或者甚至是不使用浏览器的客户。这种方式使得 Web 服务可以形成松散耦合的组件系统。

对于 Web 服务的外部使用者而言，Web 服务是一种部署在 Web 上的对象组件，它具备以下特征：

（1）良好的封装性

Web 服务是部署于网络上的对象，具备对象组件自然具有的良好封装性。Web Service 在对象的实现者（服务者）与对象的调用者（请求者）之间是一组调用接口。实现者与服务者相互分开，一个服务接口可以有多个请求者，一个请求者可以调用多个实现相同接口的服务，这种机制不仅使对象组件具有良好的封装性，提高了软件的产生能力和集成能力。

（2）分布性

Web 服务实现者与调用者可以分布在网络上的各处，可以在同一进程中，可以位于不同的机器中，也可以位于地理位置完全不同的网络中的不同机器中。对于一个应用系统而言，需要的多个服务也可以分布于网络上不同的环境中，这样，一个任务可以实现网络的分布性。

（3）使用标准协议

与一般对象相比而言，Web 服务的接口规范更加规范并且易于机器理解。首先，作为 Web 服务的对象接口所提供的功能应当使用标准的描述语言来描述（如 WSDL）；其次，由标准描述语言描述的服务接口应当能够被发现的，因此这一描述文档需要被存储在私有的或公共的注册库中。同时，使用标准描述语言描述的使用协约将不仅仅是服务界面，它将被延伸到 Web 服务的聚合、跨 Web 服务的事务、工作流等，而这些又都需要服务质量的保障。另外，对于松散耦合的系统环境安全机制非常重要，因此需要对诸如授权认证、数据完整性、事务处理的不可抵赖性等用规范方法来描述和交换。这些方面都已经有了或者正在制定一系列的公共协议，事实上，Web 服务的协议是当前计算机领域中最受人关注的话题。

（4）跨平台、跨语言

由于 Web Services 是架设在一系列通用的协议之上，其中重要的协议是基于 XML 的，这就决定了它与传统的系统集成技术不同，它可以运行于各种平台，包括典型的 Windows 和 UNIX，同时它的技术又是跨越编程语言界线的，无论是企业中广泛使用的 Java，还是桌面应用的 C++，C#，Visual Basic，都可以实现 Web 服务，并且调用者与实现者可以采用不同的编程语言。

（5）可集成能力

由于 Web 服务采取简单的、易理解的标准 Web 协议作为组件界面描述和协同描述规范，完全屏蔽了不同软件平台的差异，无论是 CORBA、DCOM 还是 RMI 都可以通过这一种标准协议进行互操作，从而实现在当前环境下各种技术的集成。在这个意义上，这种集成可以是“非紧密的”集成，是易于实现的集成。

从以上所述可以看出，Web 服务成为分布式异构系统集成的极佳手段。

1.3 空间信息 Web 服务

在现有 GIS 系统基础之上，对已有的数据及功能模块进行重新解析、包装及组合，可以实现空间信息 Web 服务。其基本要素是各个空间信息的 Web 服务，服务间数据通信采用 XML 作为数据流的格式，控制通信采用 Web 服务调用的方式。

分布性与互操作是地理信息系统的必然要求，空间信息 Web 服务可以说正是顺应了这种要求。

1.3.1 从数据共享的角度看空间信息 Web 服务

空间数据共享一直是 GIS 发展的瓶颈。地理空间数据是研究地球形成演化、探讨人类生存环境、减轻自然灾害、合理开发资源和促进社会可持续发展的重要科学依据，随着网络技术及 Web GIS 的飞速发展和应用，更加迫切要求社会各部门能够共享地理空间数据以及与之相关的资料，实现地理空间信息的全球共享。地理空间信息共享的重要性主要体现在：

1）为决策全球化问题同时提供大量的科学地理数据。

2）地理信息系统（GIS）发展的强烈要求。

3）为使用空间信息的社会各部门节省大量的人才、时间和金钱。

4）便于实现空间信息的规范化和标准化。

5）可以加强空间信息的高效管理、维护、重复利用和有效增值。

6）为数字地球建设奠定基础。

空间信息共享是指使得查询、浏览、获取、交换、使用和再加工地球上与人类生存直接或间接相关的信息能够做到方便、快捷、准确、安全和全面，包括对部分信息处理资源的自由使用。空间信息共享强调空间数据集之间的相互透明访问和信息用户对数据集的透明访问与使用，注重从空间信息的语义层次、数据模型层次和数据结构层次消除空间信息描述方法上的差异性以及表示方法上的差异性，对空间信息给出统一的描述和表示，达到空间信息本质上和形式上的共享。

空间信息共享活动涉及了三个主要概念：空间信息资源、空间信息的获取与处理和空间信息应用。而这样也就出现了三种不同的角色：空间信息提供者、空间信息处理软件和提供者——就是通常所谓的 GIS 以及用户。

空间信息共享要解决可达性、互操作性和易用性。

- 可达性是指用户能存取到数据。通过空间信息 Web 服务，用户可以通过其中提供的数据

服务，来查找、获取用户感兴趣的数据，这种数据可以分布于网络上，并由不同的服务商来提供不同区域、不同专题、不同质量的数据。

- 互操作性是指不同的 GIS 之间能互相操作、对数据能进行相同的理解。GIS 互操作的关键就是想解决空间信息异构问题。而信息具有语法和语义，可以分层次讨论信息异构问题。因此在 Internet 环境中的空间信息共享，通过空间信息 Web 服务，可以进行语法及语义差别的消除工作。在消除空间信息资源的语法差异方面，通过 Web Services，可以在数据的请求者与服务者之间用统一的数据格式，这种数据格式包括 GML 在内，它们已逐渐形成标准，并在 Web 服务中使用。在消除空间信息资源的语义差异方面，语义 Web 等方面的进展也可以对此有所帮助。

在数据共享所采用的技术方面，包括数据格式转换，通过 API 直接读取，是比较低层的方式；基于 DBMS 的集成，则可以使数据更统一，具有一定的互操作性；基于 Web Services 进行集成的系统中，数据有统一的 Schema 描述，数据之间在进行接口时有统一的协议和标准，可以有效地实现数据的共享。如图 1.2 所示。

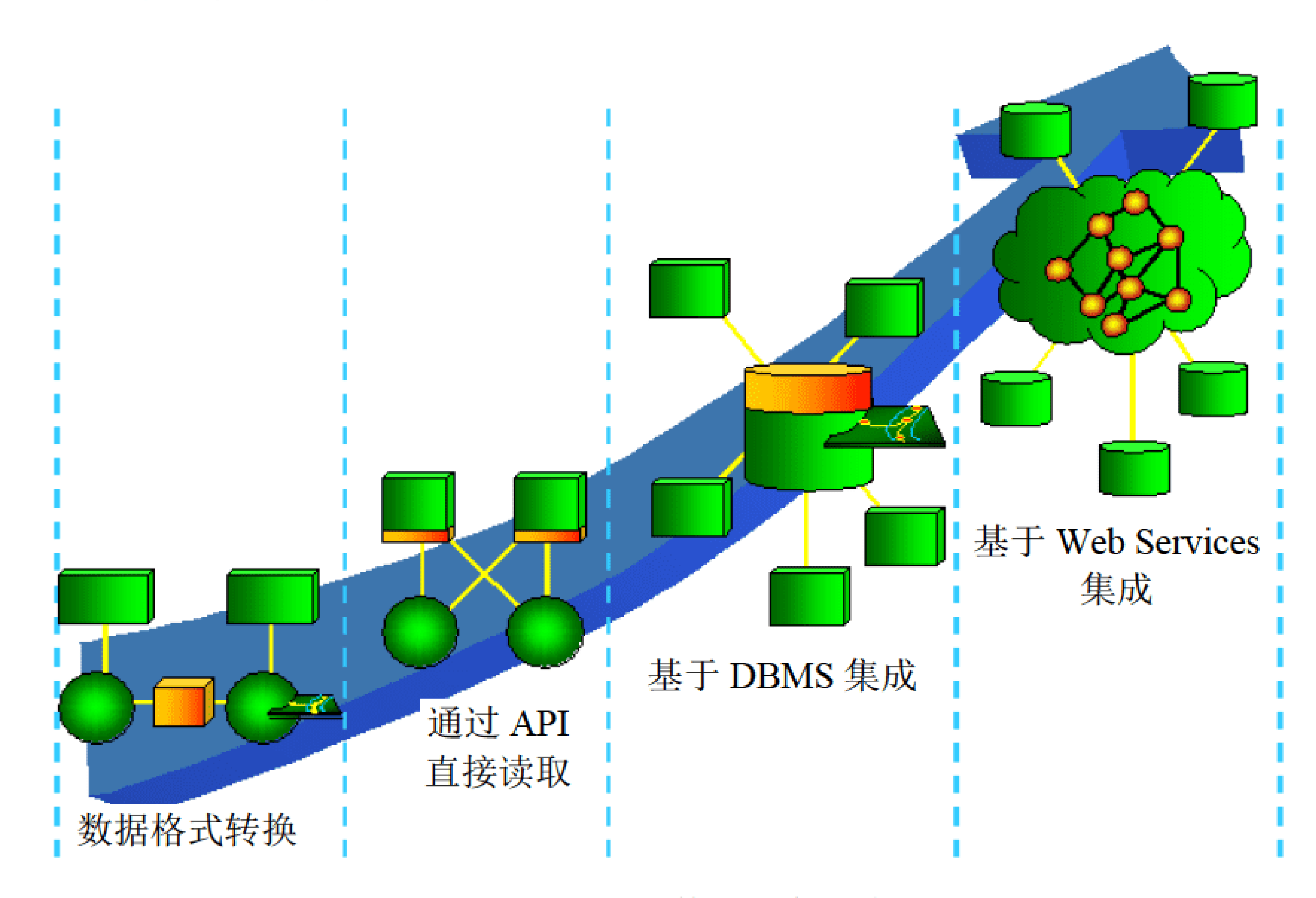


图 1.2　数据共享方式

1.3.2　从软件复用的角度看空间信息 Web 服务

GIS 技术经过长期的发展，特别是近年来的长足进步，已经积累了大量的基础软件及应用软件，这些软件功能各不相同、编程方法不同、编程接口不同、用户界面也不相同，在新的网络日益发展、软件技术飞速进步的今天，如何有效地发挥这些软件的功能，充分利用已有的软件来或软件构件，实现功能更强的软件或快速实现一个应用系统，也是摆在 GIS 软存在大量的可复用构件是有效复用的基本前提；而有效地管理大量构件，提供方便的构件存储、构件检索和构件提取等功能，则是成功复用的必要保证。Web 服务系统则为这种复用和管理提供了一种技术支撑。

首先，将现有的 GIS 系统中的功能改造成 Web 服务是可行的。由于 Web 服务采用的技术基

础是 XML，它是一种规范化的文本，易于被各种编程语言进行处理，这使得现有的 GIS 系统中的功能，进行分解、重组、规范化，从而提供 Web 服务接口是可以实施的。

其次，Web 服务中的对象，实际是网络上分布的对象。这些对象不论其内部如何实现，但这些对象之间是靠通用的接口来进行通信的，这些接口遵守各种层次的协议，如 XML、SOAP、WSDL 等，对于空间信息 Web 服务中的服务，还有与空间信息处理相关的协议，如 WFS、WCS 等等。一个系统可以方便地调用远程对象，而不论这种对象是在什么平台上，也不论这种对象是用什么编程技术来实现的，这样，通过 Web 服务的对象复用实现了更高层次的对象复用。

另外，Web 服务是服务的松散集合，可以方便地发布，并可以通过程序自动或人工进地进行查找和调用，这就给利用已有的 Web 服务带来了方便。例如，可以将现有的多个不同供应商提供的地图显示服务集合起来，再形成一个新的、面向专题的地图服务。

1.3.3 从系统集成的角度看空间信息 Web 服务

Web 服务技术在一定意义上是一种系统集成的技术。Web 服务用于 GIS 是一种更好的进行 GIS 系统集成的技术。

GIS 系统的集成技术经历了以下的一个发展过程。

（1）基于对象技术的 RPC 方法

多年以来，应用程序分布式集成计算方法的演化基本上都基于远端过程调用（RPC，Remote Procedure Call）机制。RPC 是指应用程序对运行在远端计算机上的代码进行功能调用。为实现该调用的请求，在相互协作的计算机之间，需要使用特定的协议支持来对信息数据进行打包、发送和接收。

目前基本上每种主要的对象技术均有其自己的 RPC 技术。Microsoft 组件对象模型（COM）使用 DCOM/COM+，CORBA 使用 IIOP，Java 使用 RMI。这种“分布式对象”为标志的集成分布式系统，应用于集成更广泛的分布式系统时，具有其很大的不足。由于这些应用程序接口需要“严格匹配”，并与目标系统的专用技术密切相关，所以其连接非常脆弱，很难实现真正的跨平台、异构系统的集成。由于这些相互关联的组件可能各不相同，所以开发和维护的费用非常高。由于在等待远端所访问资源的响应时，对组件进行的同步调用经常“阻塞”，所以其可扩展性也存在固有的缺陷。

（2）基于消息中间件技术的 RPC 方法

基于消息中间件技术引入了关联的方法，以便使用消息传送技术集成更大范围地域中的分布式系统，改进了对象集成技术可扩展性和可管理性。由于这样通常不会阻塞应用程序的调用，应用程序可以“发送并忘记”信息，从而使操作更有弹性和可扩展性。Microsoft、IBM、BEA 等公司都提供消息队列与事务处理中间件产品，支持分布式应用系统的集成。

基于“消息中间件”的集成技术的一个主要限制就是它的开放性。首先，即使系统最初可以接受开放数据格式，它们仍然基于专用程序很高的技术，并使用专用的接口和专用的内部数据表示方法；其次，购买这些技术、集成和后续维护昂贵，且这些技术通常需要在应用程序联接的两端同时运行特定的软件或各有一份客户许可协议，这样增加了成本、消耗时间并增加了集成的复杂性，集

成的范围也受到限制。故需要一种开放的系统集成技术，使用符合工业标准的传输协议、过程调用和数据表示方法，支持平台分布式系统的低成本集成与互操作。

（3）Web 服务方法

Web 服务方法最早起源于 XML-RPC 技术。在 1998 年早期，Userland、DevelopMentor 与 Microsoft 一起发布了 XML-RPC（http://www.xmlrpc.com）。XML-RPC 提供了一种简单的机制，使处于不同环境下的应用之间，可以通过 Internet 来进行远程过程调用。它采用 XML 作为编码标准，HTTP 为传输协议。XML-RPC 通过 HTTP 请求向 RPC 服务器提交请求，通过 HTTP 响应 Response 获得 RPC 的调用结果。这种 XML-RPC 技术突出地强调了 XML 和 HTTP 的作用，使用应用的集成可以跨平台地进行。

后来，IBM、Microsoft、SAP 等公司一道在此基础上逐步提出了 Web Services 的概念，并制定了一系列服务调用、发布、集成的协议，使 Web Services 成为新一代的系统集成方法。

空间信息 Web 服务就是将这种新的技术应用于地理信息系统，解决地理信息系统中的多源、异构、分布系统的集成问题。

1.3.4　空间信息 Web 服务的优势

与普通 Web 服务一样，空间信息 Web 服务也具有不少的优势，如：

- 分布
- 可伸缩性
- 功能全面
- 系统可维护性好
- 软件集成费用低
- 使用工业标准，所以可以方便地与其他软件及企业合作

具体来说，空间信息 Web 服务具有以下一些特色：

（1）能较好地处理数据的共享。它可以避免与具体数据格式的紧密绑定。不同的 GIS 可以选择不同的后台数据库，同时，Web Services 不但允许客户端对服务端的数据进行访问，还允许服务端对服务端的数据共享服务。一个 GIS 可以对自己的数据库（如 RDMS、SDE 或数据文件）进行优化，同时存取其他格式的数据，从而产生数据、地图或空间计算的服务，为更广大的客户端提供一个统一的数据服务环境。

（2）能建立一个标准的空间计算的基本环境框架。由于 Web Services 分布于网络上的各处节点，包括服务器、工作站、桌面客户端，以及手机、PDA 等便携产品。Web Services 的标准使得这些设备可以统一起来构成一个更广的分布计算环境。这种计算环境不仅仅是为 Internet，它可以为各种分布式计算提供有力的支持。

（3）能为各种开放式网络提供一个融合的计算环境。通过 Web Services，各种计算节点的三种角色，即客户、服务和代理，它们可以们位于 Internet、无线网络或者是局域网中。客户端可以是普通应用程序，也可以是 Web 浏览器，Java Applet，也可以是移动设备。客户端向各个服务提出请求并获得结果；服务则一直准备着提供服务；代理则为服务的发布和查找提供支持。这三种角

色相互配合，形成了一个融合的计算环境。

（4）能方便地集成。如上文所述，Web Services 之间可以互相集成，这不仅使空间信息的服务集成成为可能，同时也为 GeoWebServices 与办公软件、企业工作流、电子商务、电子政务方便地集成。例如政府部分可以将有关林业、农业、水利、环境监测等部分的地理信息集成到电子办公中去，并作为决策的依据。

由此看来，Web Services 是实现新一代 GIS（第四代 GIS）的重要手段。第四代 GIS 的目标是由以系统为中心向以数据为中心，实现空间数据共享与服务的转变，成为 OS、DBMS 之上的主要应用集成平台，Web Services 正是这种平台实现的一种基本方式。同时，Web Services 在 GIS 中的应用，同它在其他领域中的应用一样，将是革命性的变化，由于 Web Services 可以集成现有的各个平台上的 GIS 数据与服务，同时，还可以方便地与 OA、MIS、电子商务、电子政务、公众服务系统等进行集成，实现社会化 GIS、公众 GIS 等。

可以说，Web Services 将是实现空间信息基础设施的基本方式，它会在数字城市、数字地球等领域中发挥重要作用。

1.4 GIS 的 Web 服务规范

在 GIS 分布式、互操作方面，一些组织（如 OGC，UCGIS 等）正在制定相应标准和进行一些实验项目，其中 OGC（开放地理信息系统协会）在空间信息 Web Services 方面继续制定了一系列规范。

OGC 一个是由多个企业、大学、政府部门组成的非盈利性组织，最初目的是想提供一套综合的开放性接口规范，以便开发商可以根据这些规范来编写互操作组件，以满足 GIS 互操作的需求，后来就成为一个专门发展 OpenGIS 规范的机构，以制订和推进开放的空间数据互操作规范为目标。

对于 Web 服务在空间信息领域的应用，OGC 表现了极大的关注。2001 年 3 月，OGC 发出 OGC Web Services Initiative Phase 1 的技术请求，启动了 OWS 标准的开发进程。

在 OGC 制定的规范中，从规范的名称中也可以看出向 Web 服务的发展趋势，从 OGC01-065：Web Feature Server Implementation Specification 到 OGC02-058：Web Feature Service Implementation Specification（如图 1.3 所示），原先用 Server，后来用 Service，这实际上体现了从传统的 Web GIS 向 Web 服务观念的转变。

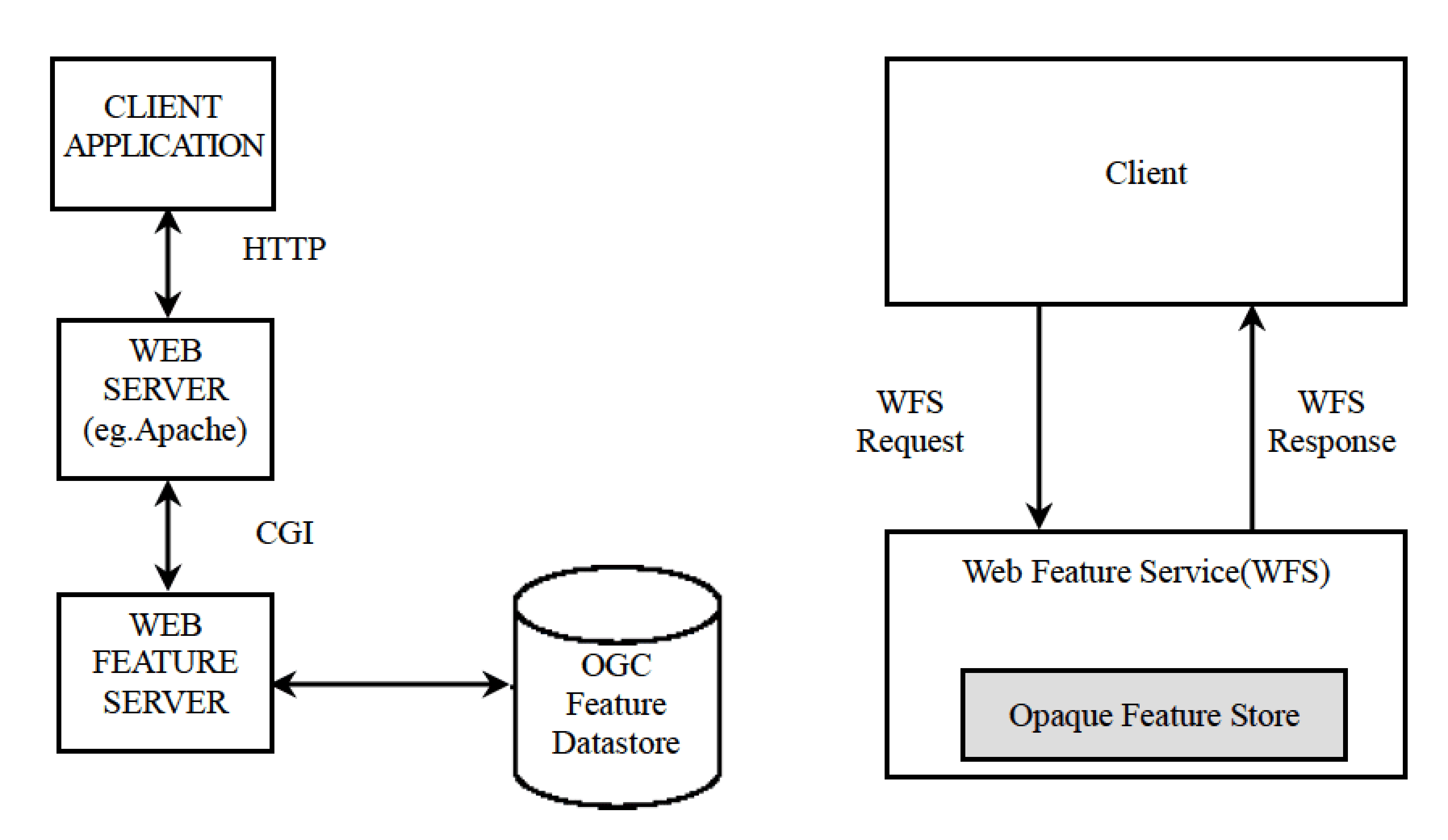


图 1.3　传统 Web GIS 与基于 Web 服务的 GIS 的对比

1.4.1　OWS 服务体系

在 OWS 服务体系中，主要的部分包括：

- 地理数据服务（Data Service）——提供对空间数据的服务，主要有 WFS（Web Feature Service，矢量数据服务），WCS（Web Coverage Service，栅格数据服务）。地理数据服务返回的结果通常是带有空间参照系的数据。
- 地图描绘服务（Portrayal Service）——提供对空间数据的描绘，主要有 WMS（Web Map Service，地图服务），其中地图可以由多个图层组合起来，每个地图可以用 SLD（Styled Layer Descriptor）来对地图进行描述。地图服务的返回结果通常是矢量图形或栅格图形。
- 过程处理服务（Processing Service）——提供地理数据的查找、索引等服务，主要有 Geocoder（地学编码服务）、Gazetteer（地名索引服务）、Coordinate Transfer Service（坐标转换服务等）。
- 发布注册服务（Registry）——提供对各种服务的注册服务，以便于服务的发现。其中包括数据类型、数据实例、服务类型、服务实例的注册服务。注册服务提供了各个注册项的登记、更新及查找服务。
- 客户端应用（Client Application）——即客户端的基本应用，如地图的显示、地图浏览以及其他一些增值服务。

1.4.2　空间信息 Web 服务的角色与功能划分

空间信息 Web 服务是一种 Web 分布式计算的框架。根据在 Web 服务中的作用，可以划分为三种基本的角色：服务的提供者、服务的请求者以及服务的中介，如图 1.4 所示。

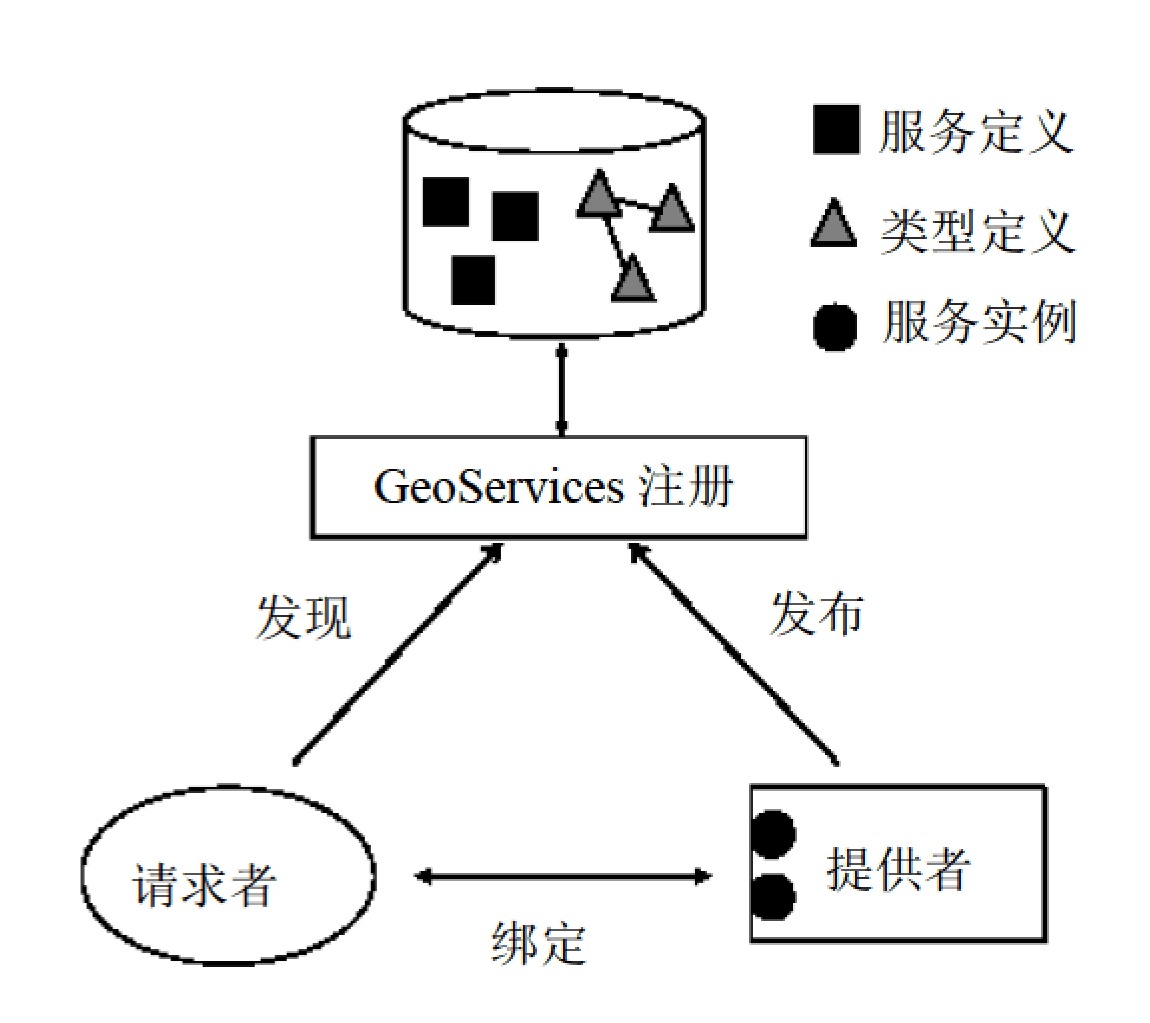


图 1.4　服务中的角色的划分

根据这三种角色，将 OGC 定义的 OWS 中的相应规范进行划分，则如图 1.5 所示：

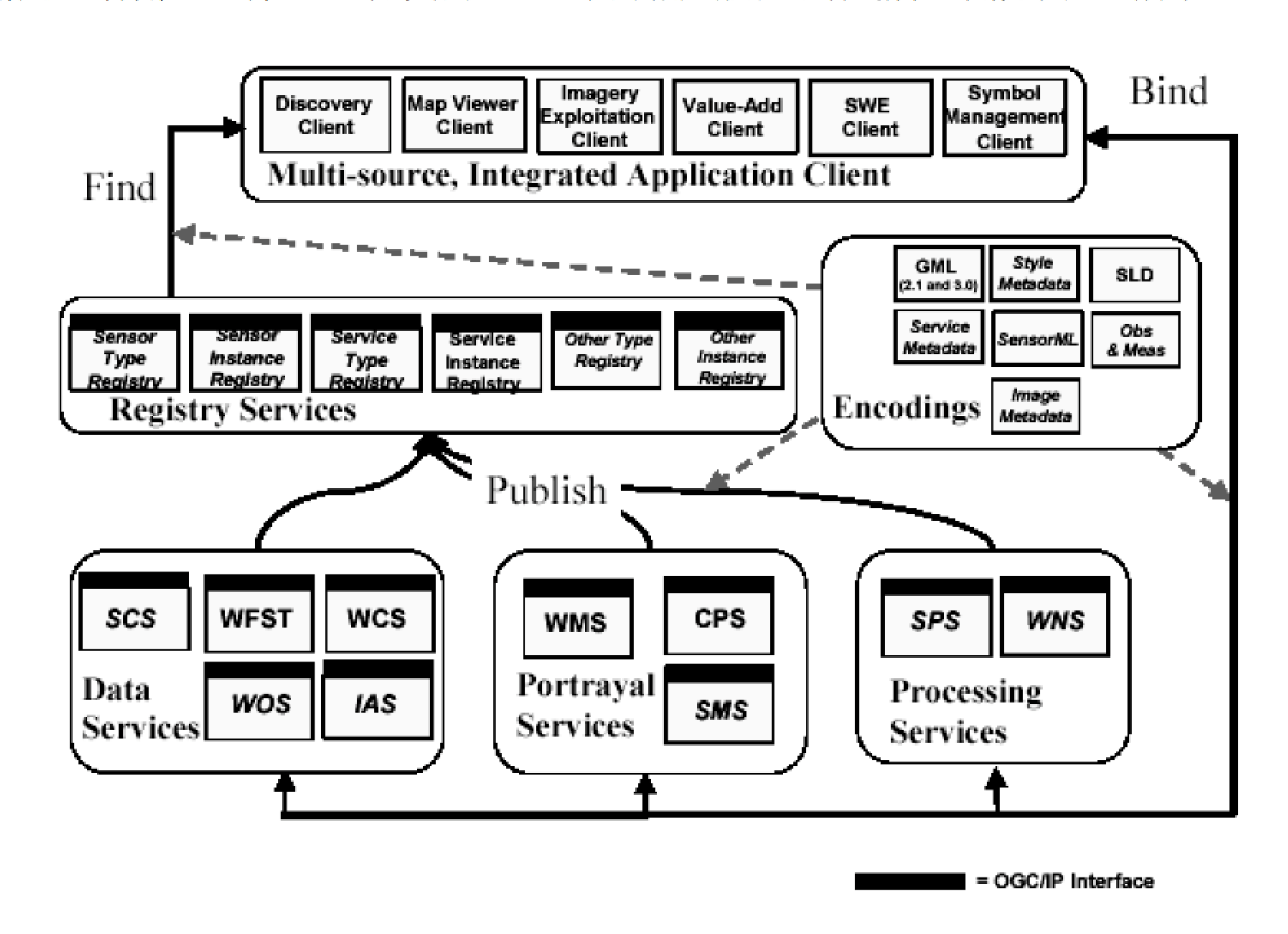


图 1.5　基于 Web Services 的 GIS 系统的体系结构

另外，对于服务与客户端的关系，如图 1.6 所示。

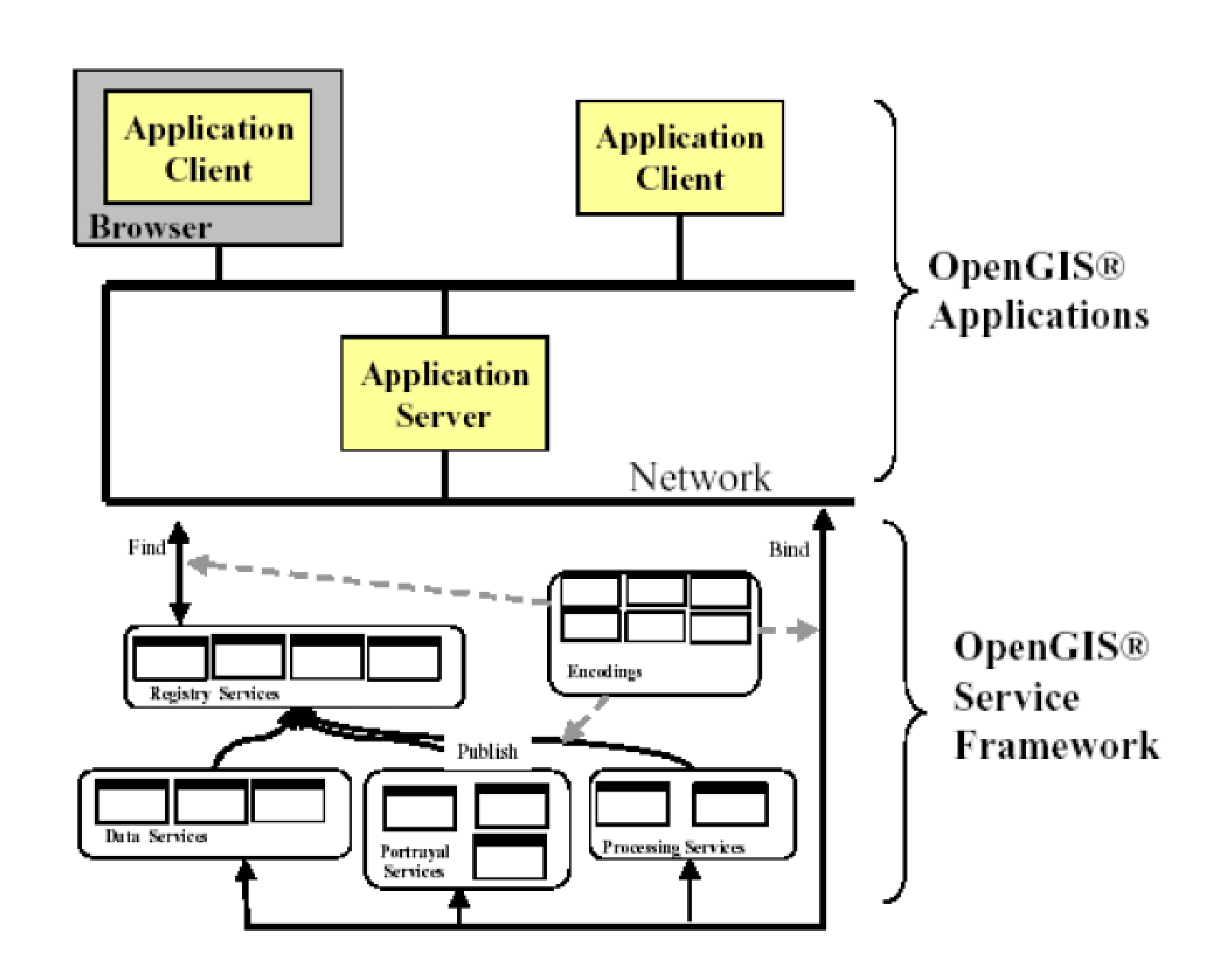


图 1.6　OpenGIS 框架下的应用客户和服务器

OpenGIS 服务框架里的应用客户又称为客户端应用程序（Client-side Applications），它应该有下面这些特点：

- 通过搜索和发现机制（使用注册服务），提供找到基于地理空间（geospatial-based）的服务和数据资源；
- 提供到影像和其他基于地理空间的应用服务和数据服务的访问；
- 与 Web/门户平台的整合；
- 以图形、图像或文字形式表现地理空间信息；
- 支持通过键盘、鼠标或其他人机界面的用户交互。

应用服务器又称为服务器端的应用客户（Server-side Application Clients），它们在网络的服务器上而不是在用户的桌面或手持设备上执行，例子包括计算密集（或 I/O 密集）的基于服务器的应用程序，如图像处理。它们的特点是：

- 由调用支撑的注册、处理、描绘和数据服务的业务逻辑组成；
- 通过 Web/门户服务器，与客户端应用程序交互。

从逻辑功能上，应用客户基本上可以分为五种类型：

（1）发现客户（Discovery Client）

收集并提交用户输入，通过注册服务查询元数据，选择一个资源（服务或数据）实例，并把该资源加到其他应用客户层里。查询屏幕可以包括三个部分以分别定义文字、关键字和地理上的搜索限制。

（2）地图查看客户（Map Viewer Client）

表现地图或地形视图，如在地图背景上渲染和显示轨迹和动态的叠加层，这些信息可以来自不同的来源。提供交互控制能力，添加和移去图层的能力，创建、选择和显示风格的能力。

（3）增值客户（Value-Add Client）

收集并提交用户输入，用用户自己的数据增加数据生产者的地理空间信息，创建新地物，更新或删除已有地物。类似地图查看客户，提供交互控制能力，添加与移去图层的能力，创建、选择和显示制图风格的能力。

（4）影像开发客户（Imagery Exploitation Client）

提供对影像的访问和查看并开发影像的工具。客户工具的例子包括平滑连续的漫游、镶嵌显示、图像闪烁、图像增强、测量、分类和注解。影像开发客户还可能与影像一起查看和创建矢量特征数据（即支持增值客户的能力）。

（5）传感器网客户（Sensor Web Client）

提供访问、管理和开发基于网络的传感器资源的方法。

1.4.3 空间信息 Web 服务的系统框架

在空间信息 Web 服务中，不仅要能存取内容，还要能存取服务。用户和软件代理可以发现、调用、组合和监控提供内容和服务的资源。不仅能提供静态的内容，而且动态控制数据、地图的产生以至相关的物理设备。

在空间信息 Web 服务中，重要的是建立一个资源描述可共享的框架，它能够发布、发现、建立及协调 Web 服务。一个空间信息 Web 服务的实现系统应具有以下功能：

- 每个服务类型可以有多个相互独立开发的实例；
- 不同种类的服务可以有相互独立的提供者；
- 在运行时能够根据服务类型、可存取的内容、服务的特点或服务质量来查找相应的服务实例；
- 可以存取时空数据所引用的元数据；
- 存取描述服务的元数据；
- 根据发现的元数据来调用相应的服务；
- 能够允许组织、协调及序列化相关的服务。

根据这些要求，可以将空间信息 Web 服务应用体系分成以下几个层次：

（1）空间数据库层；
（2）基础空间数据及空间操作服务层；
（3）元数据及目录服务层；
（4）服务管理、事务、安全控制；
（5）服务链、工作流支持层；
（6）应用集成层；
（7）具体的应用层。

各个层次的主要协议如图 1.7 所示。

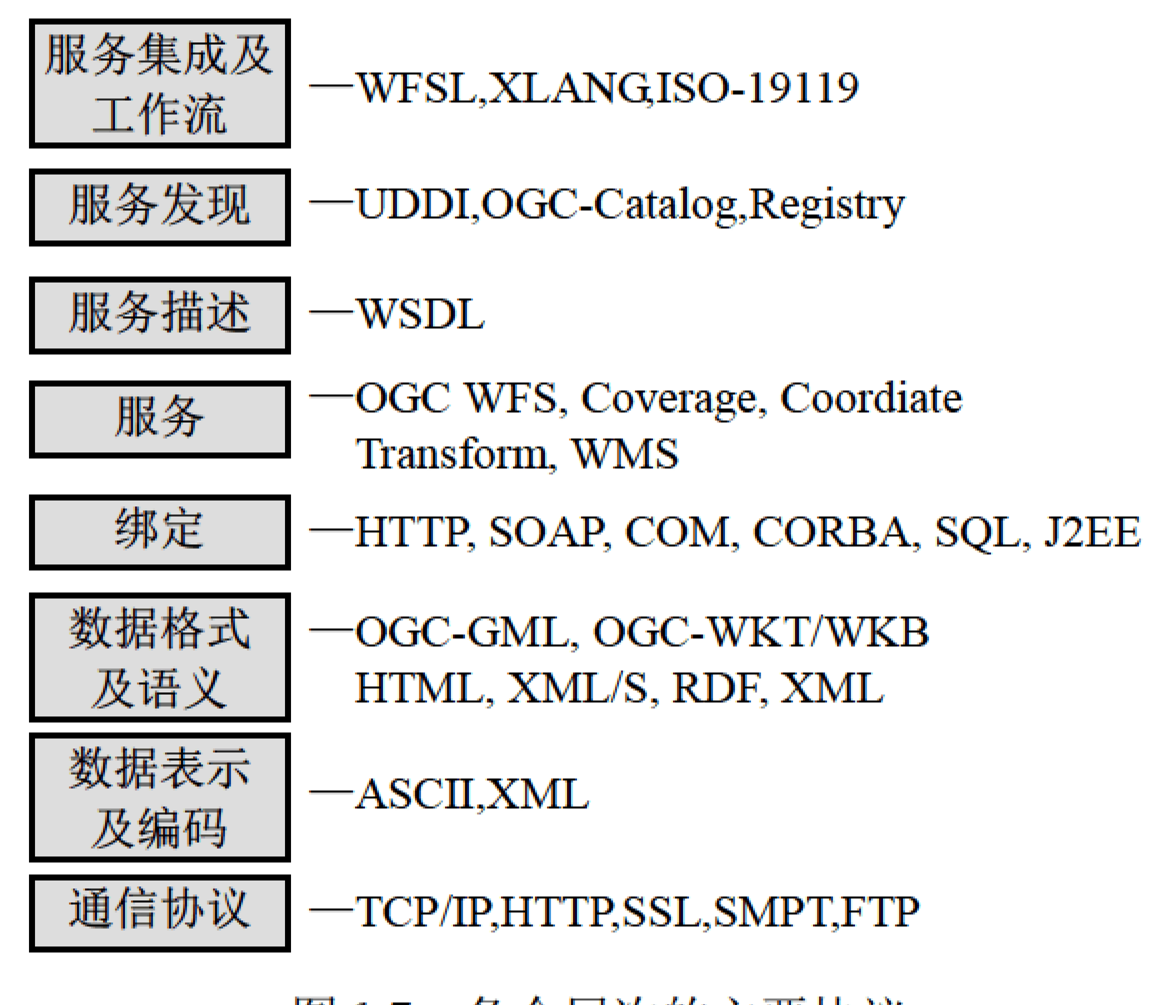


图 1.7　各个层次的主要协议

如图 1.7 所示，为了便于实现各个 Web 服务的发布、查找、绑定及调用，在可互操作的 GIS 中要基于一些已有的协议。

- 最底层是通信协议，如 TCP/IP，HTTP，SMTP，IIOP，FTP 等；在数据表示及编码层，使用 XML；在数据格式及数据 Schema 层，使用 HTML 以及 OGC 提出的表示地理要素的 GML 等。
- 绑定层主要使分布服务成为可能，绑定是在网络上连到服务端点的机制，为了使用传统的组件，在该层可以使用 COM，CORBA，J2EE 及 SQL 标准，而 HTTP 及 SOAP 是绑定到 Web Services 的基本协议。
- 在服务层，可以建立在 OGC 已定义了的简单要素的 COM、CORBA 及 SQL 规范的基础上，另外一些规范包括栅格 Coverage 规范，坐标转换，Portrayal，Gazetteer，Geocoder，Geoparser 等服务规范。
- 服务描述层用来产生用于发现服务的基本信息，包括：服务类型信息、操作、绑定规则、服务提供者的网络地址。其中 WSDL 是现阶段的标准协议。
- 服务的发现层是发布和查找服务。服务的发现层使用服务描述层的信息。UDDI 是现在 Web 服务注册和发现的标准方式；OGC 制定的 Catalog Service 规范是关于空间信息内容和服务的发现服务的。
- 最顶层是服务集成层，它用于支持决策分析、模型化、工作流、流程处理集成等。已有一些协议如 WSFL，XLANG 等，可以用来表示工作流及 Web 服务的交互与集成。

1.4.4　空间信息 Web 服务中的基础服务

为了构建基于 Web 服务的 GIS，需要定义一套基本的地理信息服务，这些基本服务实现地理信息系统的核心功能。OGC 制定了不少的地理服务的标准，较早的标准主要是基于 HTTP 对服务的调用，现在大部分的实现标准都在向 Web 服务（基于 XML）的方向转化或过渡。

1. OWS 中制定的信息服务接口

同其他的 Web 服务一样，核心的地理信息服务也用接口（Interface）来进行表达，相同的服务

都要实现相同的接口，接口之间还可以有继承关系，如图 1.8 与表 1-1 所示。

表 1-1　核心的地理信息服务的接口

服务种类	接口继承	主要接口	说明
WS（Web Service）		GetCapabilities	服务能力的查询
Web Registry Service	WS	RegisterService	注册服务信息描述
		GetDescriptor	
WMS（Web Map Service）	WS	GetMap	获取地图
		GetFeatureInfo	Feature 信息
WFS（Web Feature Service）	WS	GetFeature	获取 Feature
		DescribeFeatureType	Feature 类型描述
WCS（Web Coverage Service）	WS	GetCoverage	获取 Coverage
SLD（Styled Layer Description）	WMS	DescribeLayer	图层样式
TWFS（Transaction WFS）	WFS	Transaction	事务服务
		LockFeature	Feature 加锁
GeoCoder	WFS	GeocodeFeature	地理编码

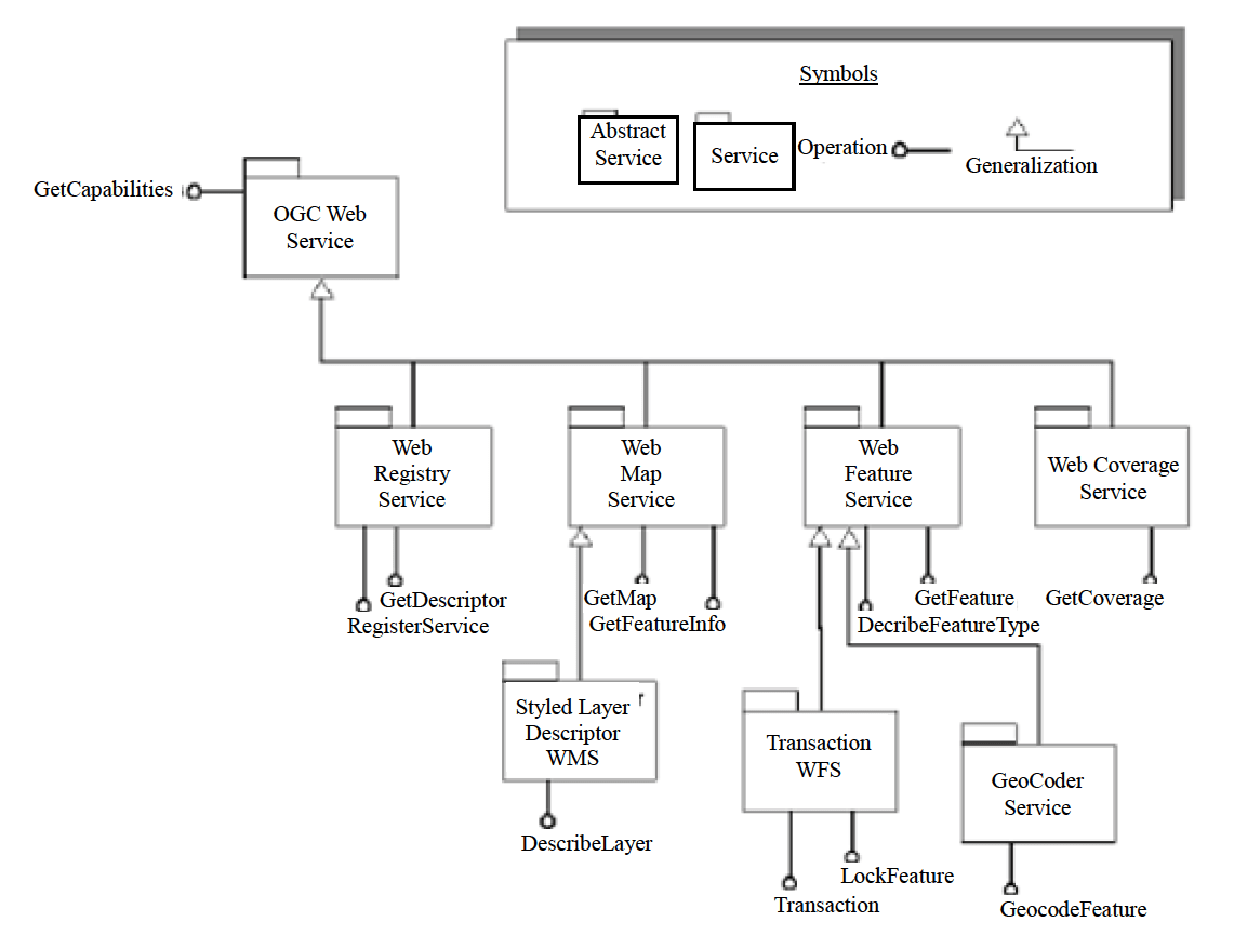


图 1.8　核心的地理信息服务接口

表 1-1 列出了不同种类的服务，分别介绍如下：

- WMS（Web Map Service）是地图服务，它将有相同的空间参照系的各个图层（包括 Feature 图层及 Coverage 图层）组合在一起。WMS 的主要接口是 GetMap，通过给定空间坐标及边界范围，可以得到相应的地图，图形的格式可以有 PNG、GIF、JPEG、TIFF 等。
- SLD（Styled Layer Descriptor）是 WMS 中的一种重要方式，SLD 是将图层中的各个要素用 Style 进行处理，即用几何对象或图标来表示图层中的各个对象，并用 XML 进行描述。基于 SLD 的 WMS 的服务结果，通常是 SVG、CGM、PS 等矢量图形格式。
- WFS（Web Feature Service）提供对地理 Feature（要素）的存取。Feature 的结果通常用基于 XML 的 GML（Geographic Markup Language）进行表达；而客户在向服务方提出的查询请求可以用基于 XML 的 Filter Encoding Specification（过滤条件编码语言）进行表达。
- WRS（Web Registry Service）为 Web 注册服务，是对元数据及服务进行的注册服务。其中，注册库中的信息可以包括：地理要素字典、服务注册库、数据模式注册库、传感器注册。
- Geocoder 服务是地理编码服务，是按地名或其他属性查询到相应的地理对象的服务，它可以提供任何地理对象相对应的网络资源的地址。

2. 常用的服务及规范

常用的服务包括注册服务、处理服务、描绘服务以及数据服务等。

（1）注册服务（Registry Services）

注册服务提供了一个分类、注册、描述、搜索、维护和访问 Web 资源信息的公共机制。注册中心有不同的角色，例如数据类型目录（地理特征、覆盖、传感器、符号等的类型）、在线数据实例目录（数据集、数据仓库、符号库等）、服务类型目录（Web Feature Server、Web Coverage Server、Web Map Server 等）、在线服务实例目录。

（2）处理服务（Processing Services）

处理服务提供对地理空间数据进行操作的服务和增值服务。给定一个或多个输入，处理服务在这些数据上执行增值处理并产生输出。处理服务可以被串联起来以完成信息生产的工作流和决策支持。这种服务序列叫做服务链（Service Chain），对每一对相邻服务来说，后面服务的执行依赖于前面服务的输出。加入到服务链后，多个服务就组合成一个互相依赖的序列以完成更大的任务。服务链要求在链内所有服务之间维持接口的一致性，这说明相对于独立的服务之间，服务链内各服务之间的耦合性要紧密的多。

典型的处理服务的例子包括：坐标转换服务（Coordinate Transformation Service）、地理编码服务（Geocoder Service）、地名词典服务（Gazetteer Service）。

（3）描绘服务（Portrayal Services）

描绘服务提供对地理空间信息进行可视化的能力。给定一个或多个输入，描绘服务会产生渲染后的输出，例如地图的制图描绘、地形的透视图、影像的注解图、特征地物在时间和空间上的动态改变等。OGC 的规范里定义了两种类型的描绘服务：

- Web Map Service（WMS）——传递以图形方式表示的地图作为对客户查询的响应。地图是地理数据的可视化表现，而不是数据本身。通常渲染的图片格式有 PNG、GIF、JPEG、

基于矢量的 SVG 和 Web CGM 等。客户在从 WMS 请求地图时需指定图层名称和图形大小及使用的空间参考系等参数。客户还可以向不同的 WMS 实例发出请求，将结果叠加以形成更丰富的地图层叠。

- Coverage Portrayal Service（CPS）——定义了对覆盖数据产生可视化图片的标准接口。覆盖数据（coverage data）是指网格化的地理空间数据，如高程数据或遥感影像。CPS 通过在瘦客户（如浏览器）上显示覆盖的视图而拓宽了覆盖数据的应用。CPS 是 WMS 的特例，请求者需要指定附加的对覆盖数据特有的参数。有时，CPS 还需要请求者指定目标 WCS（Web Coverage Service，即实际提供覆盖数据的服务）。

（4）数据服务（Data Services）

数据服务提供对数据库和数据仓库内数据集合的访问。数据服务访问的资源通常通过一个名称如标识符或地址等被引用，给定一个名称，然后数据服务找到这个资源。数据服务通常维护索引来加快通过名称或其他属性查找数据项的过程。OpenGIS 框架定义了公共的编码和接口，使多个分布式数据服务的内容以一致的方式“暴露”给框架的其他部分。OGC 的规范定义或计划定义的数据服务如下：

- Web Feature Service（WFS）——提供 GML 表示的简单地理空间特征数据，支持 INSERT、UPDATE、DELETE、QUERY 和 DISCOVERY 等操作；
- Web Coverage Service（WCS）——提供对覆盖数据的访问，数据以原始值而非渲染后的图片传递给客户，支持图像、多光谱影像、高程数据和其他科学数据；
- Sensor Collection Service（SCS）——提供一个收集和访问传感器观察数据的标准接口；
- Image Archive Service（IAS）——该服务提供对大数据量的图像和相关元数据的存储和访问，支持新图像的添加和旧图像的删除，本身可能不支持客户端定义搜索标准的查询。

（5）编码

所有 OpenGIS 框架的编码规范都采用 XML Schema 来定义。这些编码描述了特定的词汇表，用于在应用客户和服务之间、服务与服务之间封装成消息的数据传输。OGC 的规范定义或计划定义的编码如下：

- Geography Markup Language （GML）——地理置标语言（GML）是一种用于地理信息（包括地理特征的几何和属性）传输和存储的 XML 编码。如同《OpenGIS 简单特征规范》（OpenGIS Simple Feature Specification），GML 也采用了《OpenGIS 抽象规范》（OpenGIS Abstract Specification）的几何模型。与《简单特征规范》不同的是，GML 规范包含处理复杂类型属性的能力。
- XML for Image and Map Annotation（XIMA）——XIMA 定义了一个 XML 词汇表来对影像、地图和其他地理空间数据上的注记进行编码。XIMA 利用 GML 来表达这些注记的位置，并把每个注记和其描述的地理空间资源关联起来。
- Styled Layer Descriptor（SLD）——风格化图层描述符指定了一种地图风格语言的格式，以产生具有用户定义风格的地图。它满足了人或机器对地理空间数据的可视表现进行控制的需要。一个风格化的图层代表图层和风格的特别组合，图层定义了特征流，风格定义了这些特征被符号化后的风格。一个图层可被符号化成多种风格，这种关系类似于

XML 可被表现成 HTML、WML、PDF、RTF 等多种形式。

- Location Organizer Folder（LOF）——LOF 是一个 GML 应用程序模式，提供了一个结构来组织特定区域或感兴趣区的相关信息。它可以被用于各种分析应用里，如灾害分析、情报分析等。LOF 是一个信息容器，包含在空间上组织的属于某个地理区域的信息资源（空间的和非空间的）集合。许多其他的服务可以通过添加、修改或引用里面的资源来操作 LOF。
- Service Metadata——服务元数据是一个 XML 词汇表，由描述一个服务不同方面的几个单元组成。第一单元以足够的细节描述服务的接口，使一个自动化的进程能够读取描述并调用该服务所宣称的操作。第二单元描述了服务的数据内容（或其操作的数据），使服务请求者能够动态的制作请求。这个部分是可选的，取决于该服务是否包含或操作数据内容。余下的描述单元提供对特定的服务类型或服务实例专有的信息。
- Image Metadata——图像元数据是一个 XML 编码，预期会改编自 ISO/DIS 19115 元数据国际标准草案以充分描述 OGC 服务模型可以处理的所有图像类型。
- SensorML Sensor——置标语言定义了一个 XML Schema，描述传感器类型和实例的几何、动态和观测特性。
- Observations and Measurements——观察和测量定义了一个框架和 XML 编码，主要用于传感器网的环境里。

1.5　GIS 的 Web 服务实现方式

在了解 GIS 的一些基本概念后，我们来看看基于 Web Service 的 GIS 系统是如何实现对服务的请求与响应的。在此之前，先来了解一下版本号。

1.5.1　版本与流通

版本号为三段数字表示，现在 WFS 的最新版本是 1.1.0，旧版本有 0.9.1、1.0.0；WMS 的版本有：1.0、1.1、1.1.1、1.3.0。支持 WFS1.0、WMS1.1.1，其返回的 GML 版本是 2.1.2。

版本号必须出现在两个地方，一是客户端请求参数中、二是服务器 GetCapabilities 操作返回的 Capabilities XML 文档中。

需要注意的是，客户端请求的版本号应该与服务器支持的版本号匹配，否则按最近匹配原则：

（1）如果客户端请求的版本号高于服务器支持的，服务器按其支持的最高的版本号执行，相反，则按最低版本号执行。

（2）如果服务器响应的版本号高于客户端支持的，客户端会重新发送一个较低版本号的请求，相反，发送一个较高的版本。图 1.9 是以 WFS 为例的版本匹配示意图。

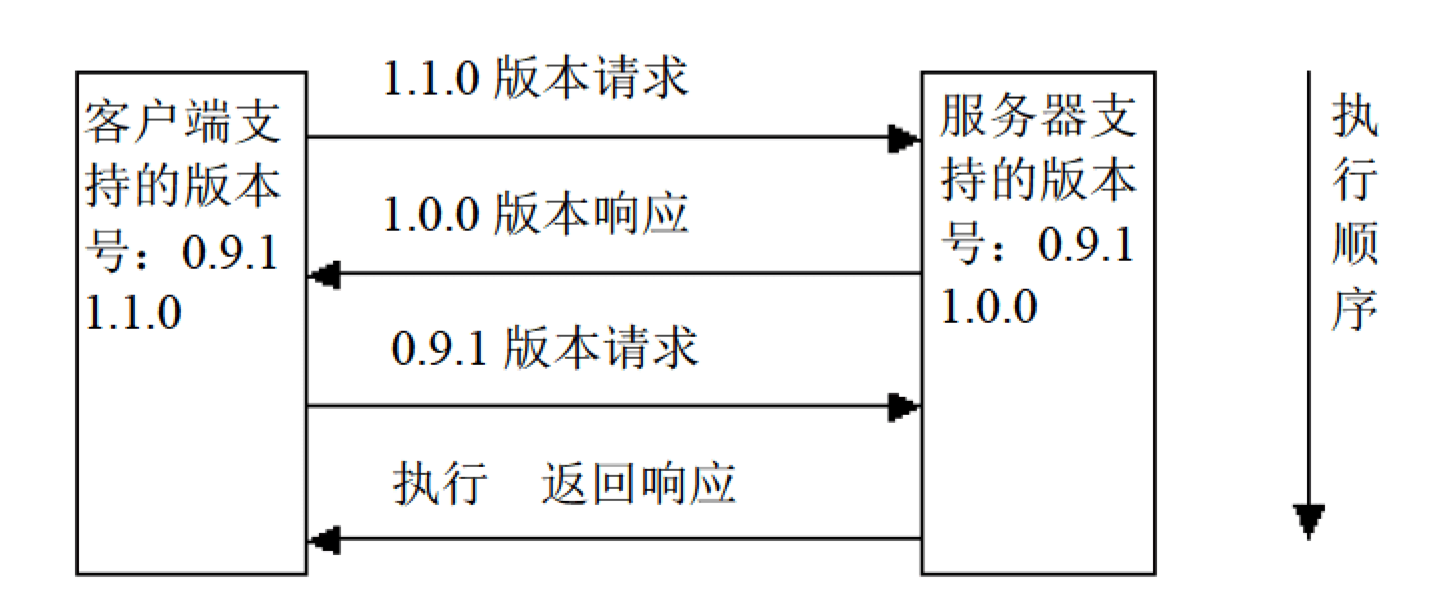


图 1.9 版本匹配示意图

1.5.2 请求规则

请求方式有两种，分别是 Get 与 Post。我们下面以访问“测绘科学数据共享服务网”（http://sms.webmap.cn/default.asp）为例来介绍 WMS 或 WFS 服务对数据进行互操作的方法。

首先需要调用 GetCapabilities 接口来获得服务的描述。在浏览器的地址栏中输入如下地址：

http://sms.webmap.cn/scripts/openserv.exe?map=/sms_ogc/sms500.map&version=1.1.1&service=WMS&request=GetCapabilities

上面是一个 Get 请求。其中 http://sms.webmap.cn/scripts/openserv.exe 是响应请求的路径，后面是参数。其中 request=GetCapabilities 表示请求的是 GetCapabilities 操作，service=WMS 表示服务是 WMS，version=1.1.1 表示使用 1.1.1 的版本，这些都是 OGC 规范中明确要求的。而 map=/sms_ogc/sms500.map 表示的是查询的是“1:500 万数字地理底图”，这是测绘科学数据共享服务网自己定义的。

上面的地址返回或打开一个 XML 格式的文件，内容如下（为了节省篇幅，删简了一些重复内容）：

```
<WMT MS Capabilities version="1.1.1">

<Service>
   <Name>OGC:WMS</Name>
   <Title>测绘科学数据共享服务系统</Title>
   <Abstract>基于地理信息互操作规范提供测绘科学数据的在线服务</Abstract>
   <KeywordList>
      <Keyword>测绘</Keyword>
      <Keyword>互操作</Keyword>
   </KeywordList>
   <ContactInformation>
      <ContactElectronicMailAddress>apsdinode@nsdi.gov.cn
</ContactElectronicMailAddress>
   </ContactInformation>
</Service>

<Capability>
```

```
      <Request>
        <GetCapabilities>
          <Format>application/vnd.ogc.wms xml</Format>
        </GetCapabilities>
        <GetMap>
          <Format>image/png</Format>
          <Format>image/tiff</Format>
          <Format>image/gif</Format>
          <Format>image/png; mode=24bit</Format>
          <Format>image/jpeg</Format>
          <Format>image/wbmp</Format>
          <DCPType>
            <HTTP>
              <Get><OnlineResource
    xmlns:xlink="http://www.w3.org/1999/xlink"
    xlink:href="http://sms.webmap.cn/scripts/mapserv.exe?map=/sms_ogc/sms500.m
ap&"/></Get>
              <Post><OnlineResource
    xmlns:xlink="http://www.w3.org/1999/xlink"
    xlink:href="http://sms.webmap.cn/scripts/mapserv.exe?map=/sms ogc/sms500.m
ap&"/></Post>
            </HTTP>
          </DCPType>
        </GetMap>
        <GetFeatureInfo>
          <Format>text/plain</Format>
          <Format>text/html</Format>
          <Format>application/vnd.ogc.gml</Format>
        </GetFeatureInfo>
        <DescribeLayer>
          <Format>text/xml</Format>
        </DescribeLayer>
        <GetLegendGraphic>
          <Format>image/png</Format>
          <Format>image/gif</Format>
        </GetLegendGraphic>
      </Request>
      <Layer>
        <Name>sms ogcservice</Name>
        <Title>测绘科学数据共享服务系统</Title>
        <SRS>EPSG:4326</SRS>
        <LatLonBoundingBox minx="106.001"
    miny="2.29468e-005"
    maxx="106.001" maxy="0.000420795" />
        <Layer>
          <Name>BOUNT</Name>
```

```
            <Title>境界</Title>
            <Abstract>境界</Abstract>
            <Layer queryable="0" opaque="1" cascaded="0">
              <Name>BOUNT500point</Name>
              <Title>1:500 万数字地理底图境界要素数据集点状要素层</Title>
              <KeywordList>
                <Keyword>境界</Keyword>
              </KeywordList>
              <SRS>EPSG:4326</SRS>
              <LatLonBoundingBox minx="69.5965"
    miny="1.8943"
    maxx="135.524" maxy="53.873" />
              <DataURL>
                <Format>text/txt</Format>
              <OnlineResource
                xmlns:xlink=http://www.w3.org/1999/xlink
    xlink:type="simple"
    xlink:href="http://sms.webmap.cn/dldata/BOUNT600.e00"/>
             </DataURL>
             <Style>
               <Name>default</Name>
               <Title>default</Title>
               <LegendURL width="20" height="10">
                 <Format>image/png</Format>
                 <OnlineResource
    xmlns:xlink="http://www.w3.org/1999/xlink"
    xlink:type="simple"
    xlink:href           =           "http://sms.webmap.cn/scripts/mapserv.exe?
map=/sms ogc/sms500.map & version=1.1.1&service=WMS &request=GetLegendGraphic &
layer=BOUNT500point & format=image/png"/>
               </LegendURL>
             </Style>
           </Layer>
         </Layer>
    </Capability>

    </WMT_MS_Capabilities>
```

其中，<Service>与</Service>之间一段的内容描述的是该服务的名称、关键词以及联系信息等。

<Capability>与</Capability>之间描述了该服务支持的操作以及包含的图层。其中<Request>与</Request>之间描述的是该服务支持的操作，从上述响应可看出该服务支持 GetCapabilities、GetMap（得到地图）、GetFeatureInfo（得到地物属性）、DescribeLayer（描述图层）与 GetLegendGraphic（得到图例）操作。<Layer>与</Layer>之间罗列了该服务所包含的所有图层数据。

在<GetMap>与</GetMap>中的 Format 列出了 GetMap 请求所支持的返回图片的格式，包括 png、tiff、gif、jpeg、wbmp 等格式，DCPType 中规定了请求的方式，上面的例子表示支持 HTTP 的 Get

与 Post 两种方式。从该响应我们可构造 GetMap 请求获取某图层或某些图层指定范围的地图。

1.5.3　响应举例

在浏览器的地址栏中输入如下请求，将得到如图 1.10 所示的一张图片。

http://sms.webmap.cn/scripts/openserv.exe?map=/sms_ogc/sms500.map&&version=1.1.1&service=WMS&request=GetMap&srs=EPSG:4326&bbox=67,3,137,55&format=image/png&layers=ROALN500arc&transparent=true



图 1.10　使用 GetMap 得到我国 1:500 万数字地理底图公路数据地图

其中，request=GetMap 表示执行 GetMap 操作，service=WMS 表示使用 WMS 服务，srs=EPSG:4326 表示使用坐标参考系统为 EPSG:4326，bbox=67,3,137,55 表示地图范围（这里以经纬度表示），layers=ROALN500arc 表示请求图层为 ROALN500arc，format=image/png 表示返回的地图图片格式为 png，transparent=true 表示透明显示。由于没有 style 参数，表示使用默认样式绘制图层。

我们再以 TerraService.net 为例来说明。TerraService 是在 TerraServer Database 基础上发展起来的 Web 服务。该数据库含有 United States Geological Survey（USGS）的 15TB 的航空数据及 1.5TB 的地形数据，TerraService 在该数据库基础上建立了一套服务，使用的平台是 Microsoft.NET，服务的主要接口是 GetMap，另外它提供了针对给定范围的地图拼接功能。

通过如下地址可得到 GetCapabilities 的 XML 文档：

http://www.terraservice.net/ogccapabilities.ashx?version=1.1.1&request=getcapabilities&service=wms

该文档指明 GetMap 请求的地址为 http://terraservice.net/ogcmap.ashx。因此可构造如下类似的GetMap 请求来得到相应范围的航空影像：

http://www.terraservice.net/ogcmap.ashx?version=1.1.1&request=GetMap&Layers=UrbanArea&Styles=&SRS=EPSG:26910&BBOX=547200,4180400,553600,4184400&width=800&height=500&format=image/jpeg&Exceptions=se_xml

该请求版本为 1.1.1，执行 GetMap 操作，请求图层为 UrbanArea，使用默认样式绘制图层，坐标参考系统使用 EPSG:26910，地图范围为 547200,4180400,553600,4184400，返回的图片宽为 800 像素，高为 500 像素，格式为 jpeg。该请求的返回结果如图 1.11 所示。



图 1.11　通过 GetMap 操作得到指定范围的航空影像

再例如，通过如下请求，可得到如图 1.12 所示的地形图：

http://www.terraservice.net/ogcmap.ashx?version=1.1.1&request=GetMap&Layers=DRG&Styles=&SRS=EPSG:26910&BBOX=524800,4166400,576000,4198400&width=800&height=500&format=image/jpeg&Exceptions=se_xml

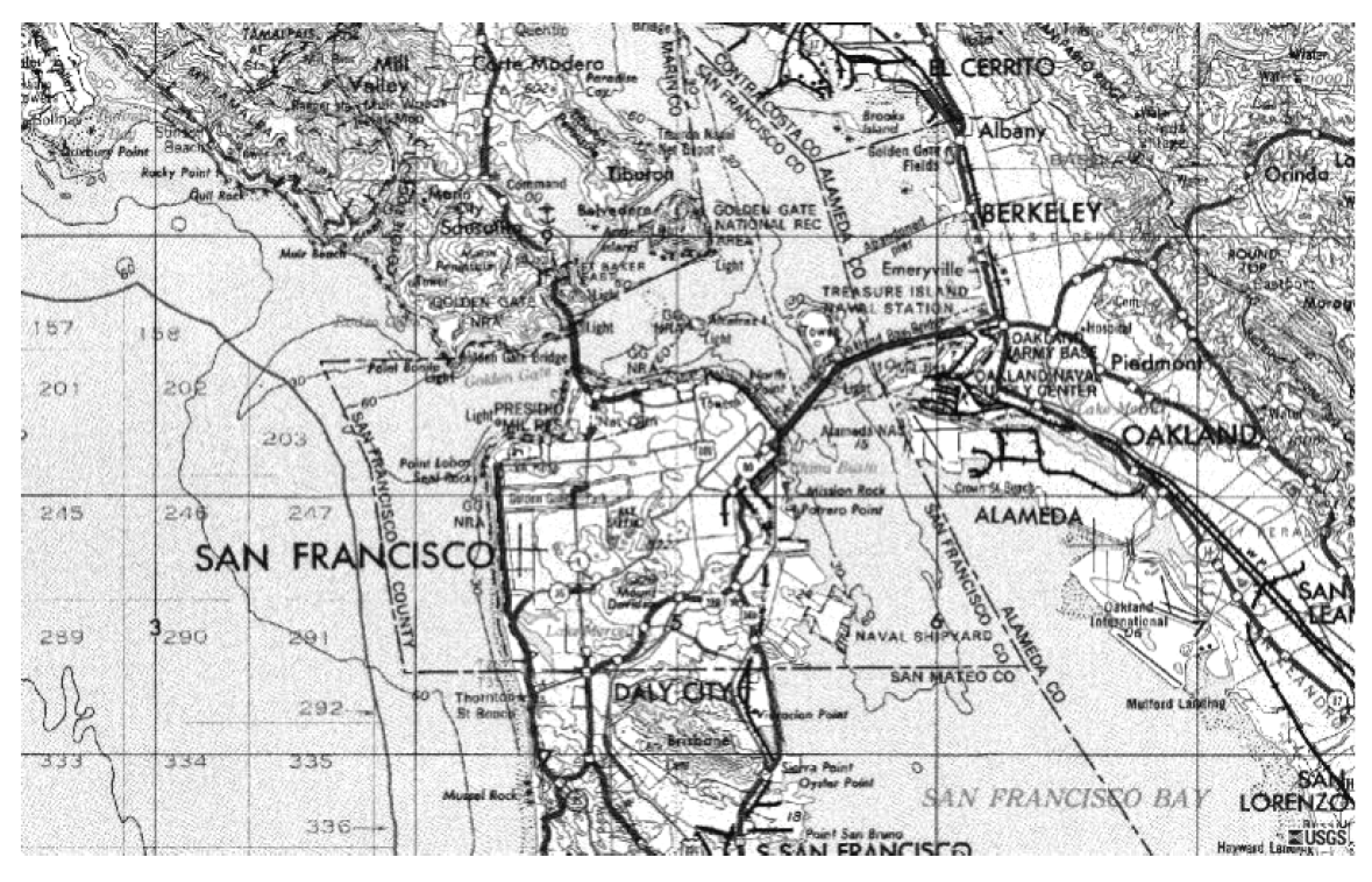
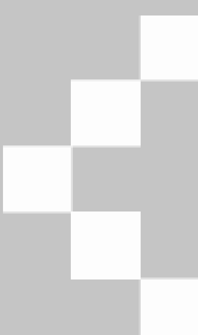


图 1.12　通过 GetMap 操作得到指定范围的数字栅格地形图

第 2 章

ArcGIS Server 9.2 简介与安装



ArcGIS Server 是一个基于 Web 的企业级 GIS 解决方案，它从 ArcGIS 9.0 版本开始加入 ESRI 产品家族。ArcGIS Server 为创建和管理基于服务器的 GIS 应用提供了一个高效的框架平台。它充分利用了 ArcGIS 的核心组件库 ArcObjects（简称 AO），并且基于工业标准提供 Web GIS 服务。ArcGIS Server 将两项功能强大的技术——GIS 和网络技术结合在一起，GIS 擅长与空间相关的分析和处理，网络技术则提供全球互联，促进信息共享。这两项技术协同工作，相得益彰。

在本章的内容中你将了解：

- 2.1 ArcGIS Server 9.2 主要功能
- 2.2 ArcGIS Server 的产品级别分类
- 2.3 ArcGIS Server 9.2 系统组成部分
- 2.4 ArcGIS Server 包含的主要技术
- 2.5 ArcGIS Server 9.2 安装

与过去的 Web GIS 产品相比，ArcGIS Server 不仅具备发布地图服务的功能，而且还能提供灵活的编辑和强大的分析能力，这对于 Web GIS 发展可以说是具备里程碑意义的。由于 ArcGIS Server 基于强大的核心组件库 ArcObjects 搭建，并且以主流的网络技术作为其通信手段，所以它具有许多优势和特点，例如：

（1）集中式管理带来成本的降低。无论是从数据的维护和管理上还是从系统升级上来说，都只需要在服务器端进行集中的处理，而无需在每一个终端用户上做大量的维护工作，这不但极大地节约投入的时间成本和人力资源，而且有利于提高数据的一致性。

（2）瘦客户端也可以享受到高级的 GIS 服务。过去只能在庞大的桌面软件上才能实现的高级 GIS 功能的时代终止于 ArcGIS Server。通过 ArcGIS Server 搭建的企业 GIS 服务使得客户端通过网页浏览器即可实现高级的 GIS 功能。

（3）使 Web GIS 具备了灵活的数据编辑和高级的 GIS 分析能力。用户在野外作业时可以通过移动设备直接对服务器端的数据库进行维护和更新，大大减少了回到室内后的重复工作量，为野外调绘和勘察提供了极大的便利。另外，ArcGIS Server 可以实现网络分析和 3D 分析等高级的空间分析功能。

（4）支持大量的并发访问，具有负载均衡能力。ArcGIS Server 采用分布式组件技术，可以将大量的并发访问均衡地分配到多个服务器上，可以大幅度降低响应时间，提高并发访问量。

（5）可以根据工业标准很好的与其他的企业系统整合，进行协同工作，为企业经营管理提供支持。例如：GIS 和客户关系管理系统（CRM）整合，发挥 GIS 的独特优势，使得企业可以打破地域的限制，更好的进行客户资源的开发，提供客户满意的产品和服务。

（6）ArcGIS Server 的出现使得我们可以利用主流的网络技术（例如.Net 和 Java）来定制适合自身需要的网络 GIS 解决方案，具有更大的可伸缩性来满足多样化的企业需求。

2.1　ArcGIS Server 9.2 主要功能

ArcGIS Server 是功能强大的基于服务器的 GIS 产品，用于构建集中管理的、支持多用户的、具备高级 GIS 功能的企业级 GIS 应用与服务，如空间数据管理、二维三维地图可视化、数据编辑、空间分析等即拿即用的应用和类型丰富的服务。ArcGIS Server 是用户创建工作组、部门和企业级 GIS 应用的平台，通过 ArcGIS Server 创建集中管理的、支持多用户的、提供丰富 GIS 功能、并且满足工业标准 GIS 应用。ArcGIS Server 提供广泛的基于 Web 的 GIS 服务，以支持在分布式环境下实现地理数据管理、制图、地理处理、空间分析、编辑和其他的 GIS 功能。其主要功能包括：

- 提供通用的框架在企业内部建立和分发 GIS 应用；
- 提供操作简单、易于配置的 Web 应用；
- 提供广泛的基于 Web 的空间数据获取功能；
- 提供通用的 GIS 数据管理框架；
- 支持在线的空间数据编辑和专业分析；
- 支持二维三维地图可视化；
- 除标准浏览器外，还支持 ArcGIS Desktop 和 ArcGIS Explorer 等桌面客户端；
- 可以集成多种 GIS 服务；

- 支持标准的 WMS、WFS;
- 提供配置、发布和优化 GIS 服务器的管理工具;
- 提供.NET 和 Java 软件开发工具包;
- 为移动客户提供应用开发框架（只包含的.NET 开发工具包中）。

实际上，ArcGIS Server 功能可以分为两大类。

一类是它本身就是一个完整的、集成的服务器端地理信息系统，提供了许多即拿即用的空间数据、空间分析、制图等的应用与 Web 服务， 这些服务可供 ESRI 产品家族中的 ArcGIS 桌面应用程序（ArcMap、ArcCatalogue 等）、ArcGIS Explorer（免费产品，可自由下载）使用，也可供支持相关标准协议的应用使用，还可供嵌入到自行开发的应用系统中。如图 2.1 所示。

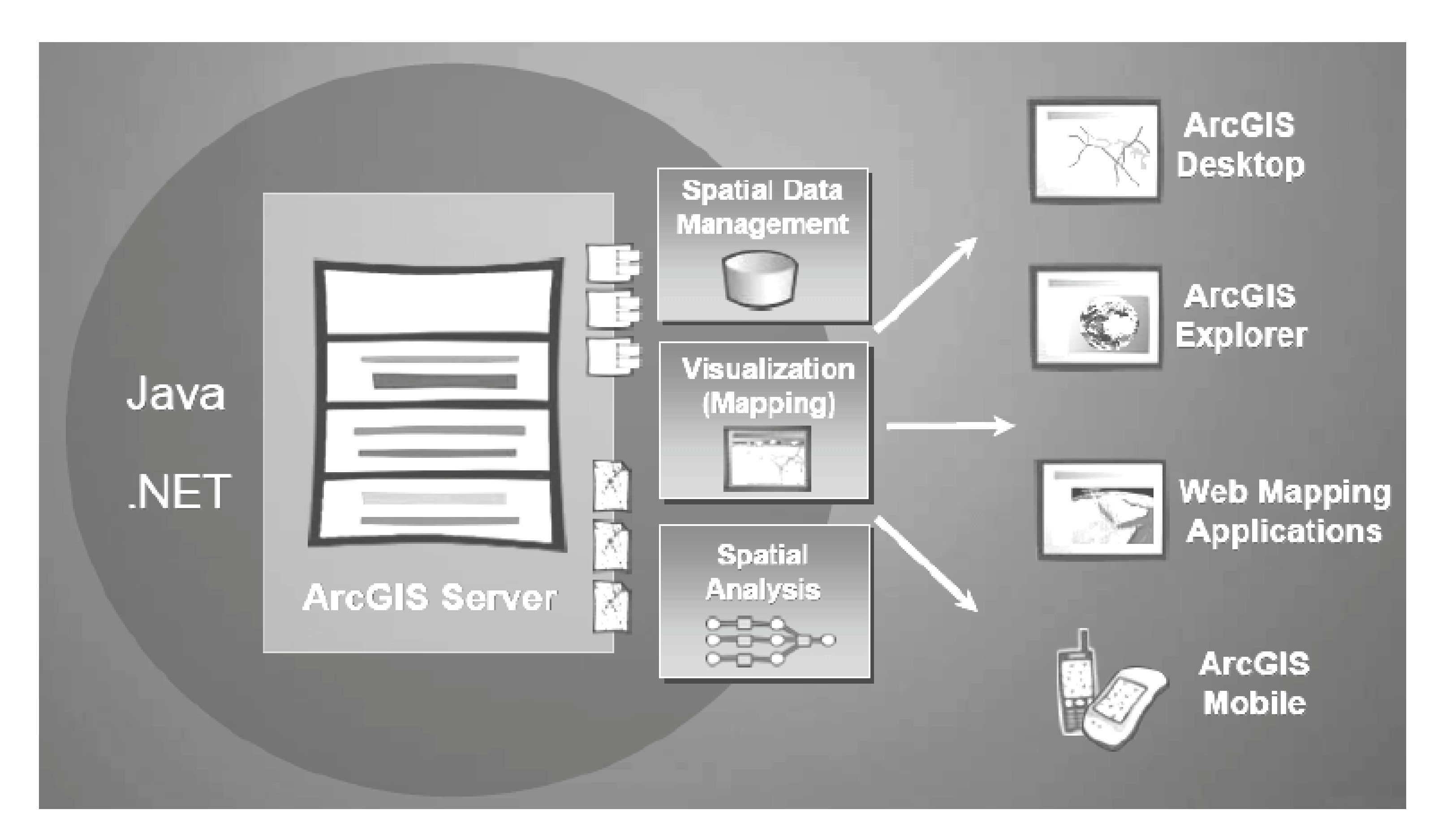


图 2.1 ArcGIS Server 9.2 功能（服务视图）

Web GIS

另一类是对再次开发支持的功能，即 ArcGIS Server 为.NET 与 Java 分别提供了一套应用程序开发框架（Application Development Framework，简称 ADF），如图 2.2 所示。应用程序开发框架可以帮助用户创建和配置.NET 或者 Java 桌面和网络应用，它们在 GIS Server 中运行时需要调用 ArcObjects。ADF 包含一个软件开发工具包（SDK），其中有软件对象、网络控件、网络应用模板、开发者帮助、和源代码范例。还包含了一个网络应用程序运行时（Web Application Run Time），这样无需在自己的服务器上安装 ArcObjects 就可以配置应用网络应用程序。

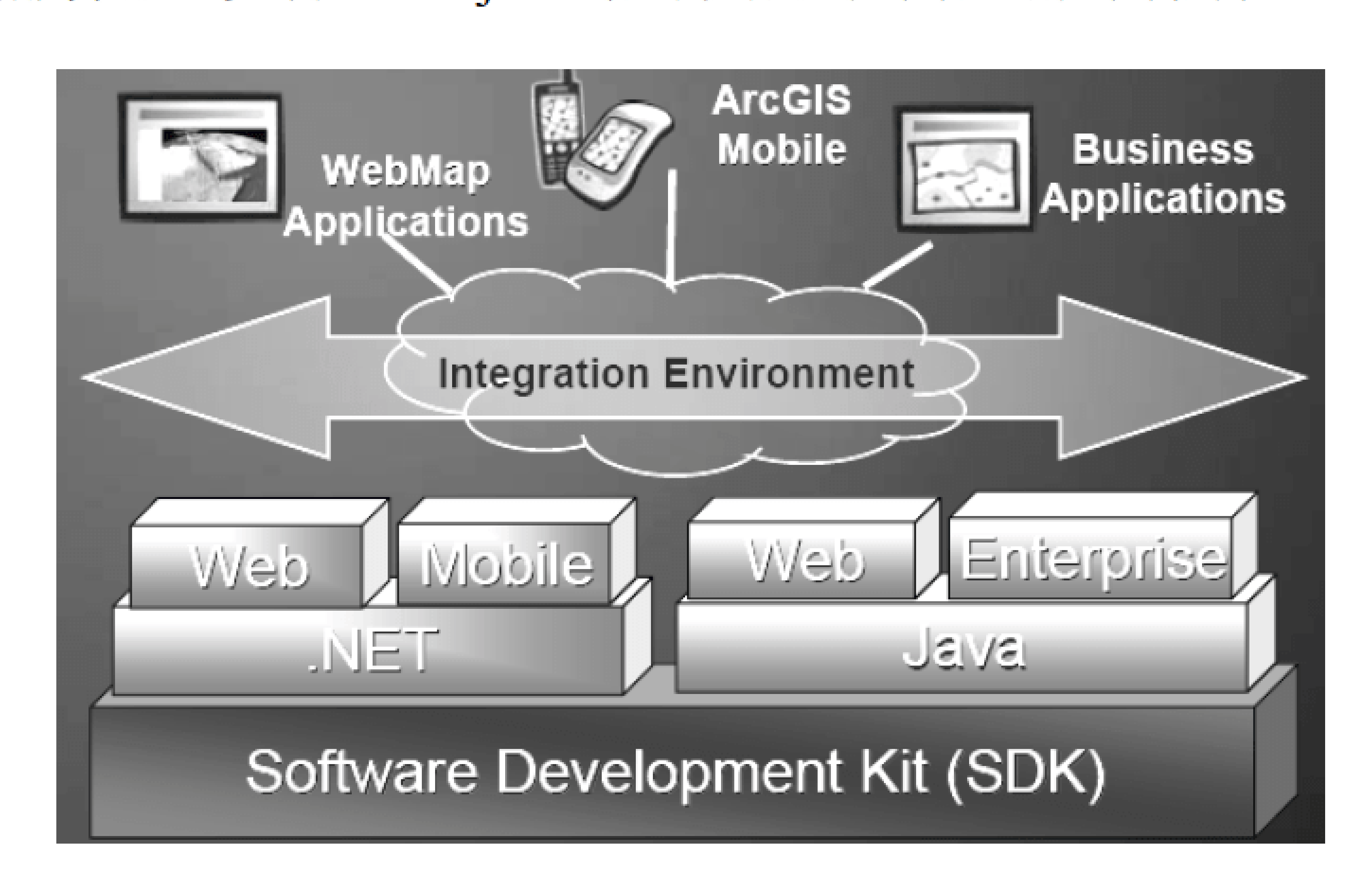


图 2.2 ArcGIS Server 9.2 功能（应用程序开发框架视图）

按这两类分类的 ArcGIS Server 功能如表 2-1 所示。

表 2-1　ArcGIS Server 功能

即拿即用的 ArcGIS Web 服务	ArcGIS Server 开发支持功能
二维与三维地图服务	基于 Ajax 的 Web 应用
地理编码服务	可以使用 Microsoft Windows 移动开发技术的开发移动应用
用于工作流自动化和分析的空间处理服务	可以通过 ArcSDE 和 SQL API 操作空间数据库
空间数据管理服务	通过 SOAP 定制 Web 服务
支持 SOAP、OGC WMS 和 KML	与面向服务的体系结构（SOA）集成

从图 2.2 可以看出，ArcGIS Server 为.NET 与 Java 分别提供 Web 服务与开发支持，这两类版本的功能有些细微的差别。

（1）ArcGIS Server .NET 功能

ArcGIS Server .NET 地理 Web 服务功能包括：

- ❑ 在各种指定的比例尺下，使用一个预处理的 2D 地图高速缓冲存储器来更快显示地图；
- ❑ 执行空间处理的工具和模型；
- ❑ 创建和发布 ArcGIS Explorer 地图；
- ❑ 基于空间处理来创建和发布 ArcGIS Explorer 任务；
- ❑ 执行 geodatabase 的复制，同步和检出；
- ❑ 以 KML 形式为 Google Earth 动态的提供地图；
- ❑ 以 WMS 服务形式为 WMS 浏览器动态的提供地图；
- ❑ 使用网络服务发布器发布标准的 SOAP 网络服务；
- ❑ 使用预先构建的 3DGlobe 数据高速缓冲存储器为 ArcGIS 桌面客户端（ArcGlobe，ArcGIS Explorer，ArcGlobe 控件，ArcReader）提供 3D Globe 数据，从而获取快速的 3D 性能；
- ❑ 使用全新的数据互操作扩展模块来通过空间处理执行提取，转换和装载（ETL）服务；
- ❑ 用 Maplex 对所需的动态地图和地图进行标注。

ArcGIS Server .NET 应用程序的构建功能包括：

- ❑ 增强的高性能 Web 地图浏览器工具，包括无缝漫游、动态缩放、地图提示、键盘快捷键；
- ❑ 非开发人员通过构建网站工具可以快速地创建 Web 应用；
- ❑ 使用全新的多服务构架，在一个网络应用程序中集成多种 GIS 服务（如 ArcGIS Server、ArcIMS、WMS 与 ArcWeb 服务等）；
- ❑ 与 Visual Studio 2005 无缝集成，包括 WEB 控件和应用模板，用于构建面向 GIS 任务的 Web 应用；
- ❑ 用全新的移动 ADF 开发轻量级的，易于配置的应用程序，供桌面和移动设备使用；
- ❑ 使用 Server 对象扩展来客户化功能，从而扩展 server 对象；
- ❑ 为服务器应用程序获取故障恢复（failover）和轮转调度（round-robin）支持。

（2）ArcGIS Server Java 功能

ArcGIS Server Java 地理 Web 服务功能包括：

- ❑ 在各种指定的比例尺下，使用一个预处理的 2D 地图高速缓冲存储器来更快显示地图；
- ❑ 执行空间处理的工具和模型；
- ❑ 创建和发布 ArcGIS Explorer 地图；
- ❑ 基于空间处理来创建和发布 ArcGIS Explorer 任务；
- ❑ 执行 geodatabase 的复制，同步，还有检出；
- ❑ 以 KML 形式为 Google Earth 动态的提供地图；
- ❑ 以 WMS 服务形式为 WMS 浏览器动态的提供地图；
- ❑ 使用预先构建的 3DGlobe 数据高速缓冲存储器为 ArcGIS 桌面客户端（ArcGlobe，ArcGIS Explorer，ArcGlobe 控件，ArcReader）提供 3D Globe 数据，从而获取快速的 3D 性能；
- ❑ 用 Maplex 对所需的动态地图和地图进行标注；
- ❑ 用一个全新的基于网络的 Java 管理工具来管理你的服务器；
- ❑ 通过标准的 UNIX 认证（必须要 Windows 认证）来验证用户；
- ❑ 使用全新的数据互操作扩展模块来通过空间处理创建 ETL 服务。

ArcGIS Server Java 应用程序的构建功能包括：

- ❑ 增强的高性能 Web 地图浏览器工具，包括无缝漫游、动态缩放、地图提示、键盘快捷键；
- ❑ 非开发人员通过构建网站工具可以快速地创建 web 应用；
- ❑ 使用全新的多服务构架，在一个 Web 应用程序中集成多种 GIS 服务（如 ArcGIS Server、ArcIMS、WMS 与 ArcWeb 服务等）；
- ❑ 与 Eclipse 和 Sun Creator 无缝集成，包括 web 控件和应用模板，用于构建面向 GIS 任务的 web 应用；
- ❑ 使用了新的企业级 ADF 中包含的细粒的空间企业级 JavaBeans，用于开发企业级 Web 应用程序；
- ❑ 使用改良的 Java API；
- ❑ 为服务器应用程序获取故障移交（failover）和轮转调度（round-robin）支持。

2.2 ArcGIS Server 的产品级别分类

为了满足工作组级、部门级、以及企业级的需求，ArcGIS Server 依据其功能和服务器规模差异，提供了一个可伸缩的产品线。ArcGIS Server 从服务器规模上分为工作组级和企业级两个级别；又从功能上分为基础版、标准版与高级版三个级别的版本。因此 ArcGIS Server 包括了 6 个不同级别的产品。

2.2.1　按功能分级

从功能上 ArcGIS Server 分为基础版、标准版与高级版三个级别的版本。

- 基础版为用户提供用于空间数据管理的 GIS 服务器。它主要利用 ArcSDE 技术来组织和管理地理数据集。
- 标准版为用户提供用于空间数据管理和可视化（制图）的 GIS 服务器。它的功能包括二维制图、三维渲染服务和一系列相关功能，如地理编码、地名辞典和路径分析。应用开发人员可以通过访问组件（对象、Web 控件和服务）来构建 Java 和.NET 框架下的解决方案。ArcGIS Server 标准版包含所有基础版的功能。
- 高级版为用户提供用于空间数据管理、制图、3D 可视化和基于浏览器的编辑、地理处理、空间分析、建模等功能。高级版包含所有基础版和标准版的功能。对于开发人员而言，高级版含有多层组件用于为桌面、移动客户端、智能客户端、网络浏览器和企业模式构建和部署 Java 和.NET 的应用和服务。

三个版本的功能比较如表 2-2 所示。

表 2-2　ArcGIS Server 基础版、标准版与高级版功能的比较

功能	基础版	标准版	高级版
多用户空间数据库	有	有	有
基于 Web 的复制	有	有	有
Web 制图	无	有	有
三维服务	无	有	有
地理处理	无	有限	有
机遇 Web 的编辑	无	无	有
移动应用	无	无	有

2.2.2　按规模分级

ArcGIS Server 工作组级（Workgroup）的 ArcGISServer 仅能运行于单台单 CPU socket（单核或双核）的机器上，并且使用 Microsoft SQL Server Express 数据库引擎支持空间数据库。

ArcGISServer 企业级（Enterprise）的 ArcGISServer 可以运行在一台或多台机器上，并且每台机器可以有多于 2 个 CPU socket。ArcGIS Server 企业级包 ArcSDE，用户需要自行提供 DBMS（SQLServer、IBM DB2、Informix 或 Oracle）。

2.2.3 可选扩展模块

ArcGIS Server 中还有一系列的可选扩展，可用来补充其核心系统的能力。

1. ArcGIS Server Spatial

ArcGIS Server Spatial 扩展提供一套强大的功能，用于创建、查询和分析基于像素的栅格数据。在 ArcGIS Server 中使用 Spatial 扩展可以从现有数据推导出有价值的信息、确认空间关系、找到适宜位置、计算旅行代价表面以及执行大量的栅格地理处理操作。用 ArcGIS Spatial Analyst 扩展创建的模型和工具可以利用这个扩展发布为 Web 服务。

2. ArcGIS Server 3D

ArcGIS Server 3D 扩展提供了一套 3D GIS 功能用于创建和分析表面。3D 扩展添加了一些基于 3D 和地形的地理处理操作，这些操作可以发布为 Web 服务。

3. ArcGIS Ser ver Network

ArcGIS Server Network 扩展提供基于网络的空间分析能力，包括路径、旅行方向、最近设施和服务区域分析。开发人员可以使用它构建和部署网络应用。

4. ArcGIS Server Data Interoperability

ArcGIS Server Data Interoperability 扩展可以方便使用和分发不同格式的数据。使用 Data Interoperability 扩展可以直接读取超过 70 种空间数据格式，导出为数十种空间数据格式。使用 ArcToolbox 中的 Quick Import Quick Export 工具，可以在各种数据格式之间快速转换。使用 Workbench 的语意翻译引擎和 Spatial ETL 工具，可以执行高级的数据转换。ETL（Extract-Transform-Load）是用于转换数据的工具，它可以在多种计算环境间轻易迁移。使用 ArcGIS Data Interoperability 扩展创建的特殊格式和翻译器可以用在 ArcGIS Server Web 服务和地理处理服务中，以支持自动和开放的数据交换。

2.3 ArcGIS Server 9.2 系统组成部分

ArcGIS Server 是一个分布式系统，由分布在多台机器上的各个角色协同工作。基于 ArcGIS Server 搭建的 Web GIS 解决方案支持多种类型的客户端，包括 ArcGIS 桌面产品、ArcGIS Engine 应用与 Web 浏览器。利用 ArcGIS Server 搭建的 Web GIS 架构如图 2.3 所示。

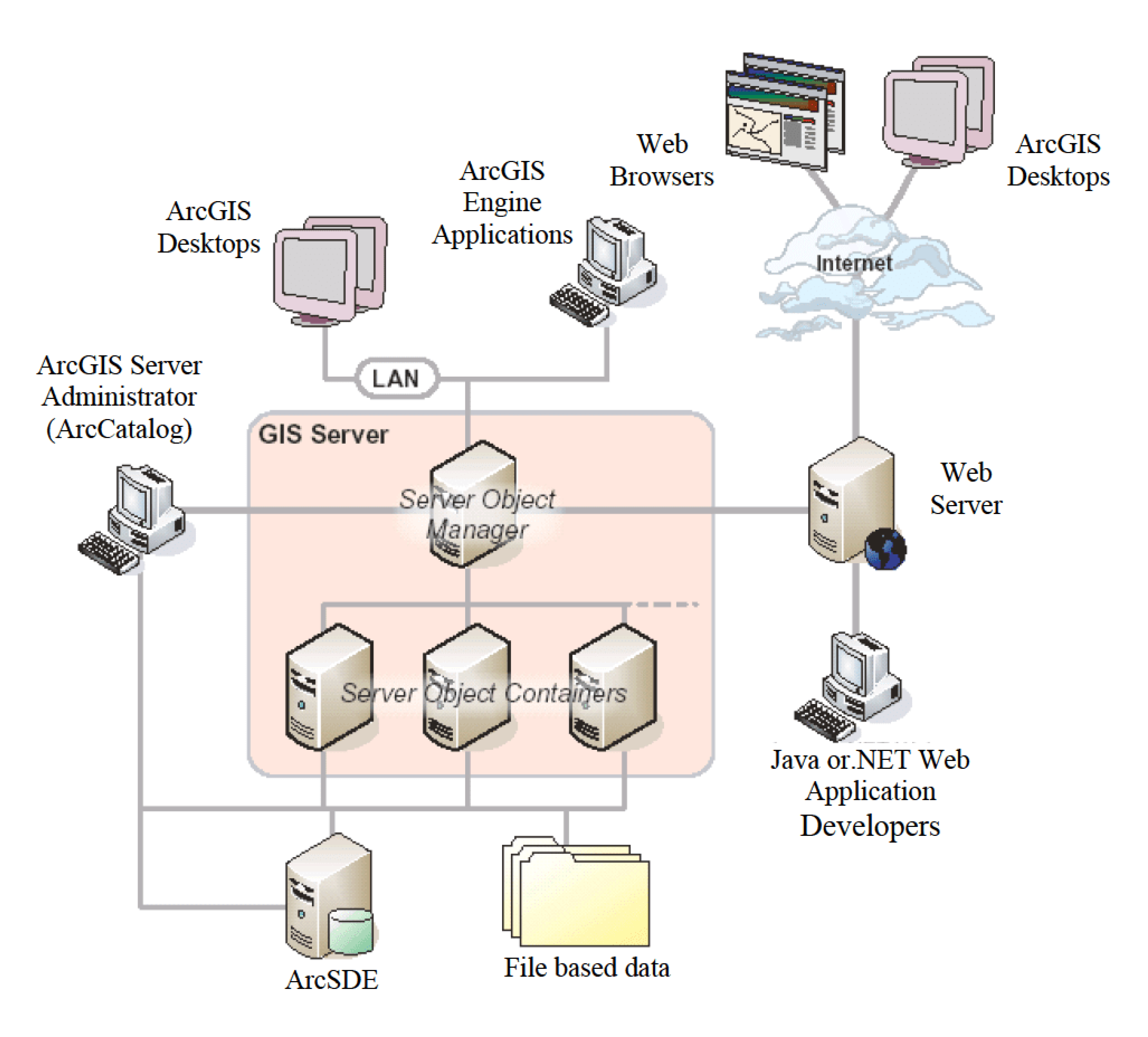


图 2.3　利用 ArcGIS Server 搭建的 Web GIS 架构

2.3.1　GIS 服务器

GIS Server 宿主了所有的 GIS 的资源，比如地图、3 维场景等，并且把它们作为服务发布到客户端。

GIS Server 本身包含了两个部分，SOM（Server Object Manager，服务器对象管理器）和 SOCs（Server Object Containers，服务器对象容器）。SOM 管理着运行在服务器上的服务，当客户端请求一个服务的时候，SOM 负责分配一个服务给客户端使用，它起到一个负载均衡的作用，一个管理服务的作用。 SOM 连接着一个或者多个 SOC，真正的服务都是在 SOC 的机器上运行着的，也就是说 SOC 才是真正的服务的宿主，所有的客户端的请求通过 SOM 分配以后都是由某个 SOC 来负责完成的。当然根据配置的不同，SOM 和 SOC 可以分配到不同的机器上，也可以由多个 SOC 的机器，图 2.3 即显示的是一个 SOM 机器连接着三个 SOC 机器。

2.3.2　Web 服务器

Web 服务器是运行 Web 应用程序或 Web Service 的机器。这里的 Web 应用程序或 Web Service

通过访问 GIS Server 并调用 GIS Server 的对象来实现 GIS 功能，然后把结果返回给客户端。

2.3.3 客户端

通过 Http 方式或者局域网方式连接到 ArcGIS Server 的各种客户端：浏览器，C/S 程序或者移动设备等。

2.3.4 数据服务器

包含 GIS Server 上所发布服务的 GIS 资源，可以是 mxd 文档、geodatabase、toolbox 等。

2.3.5 管理工具

Manager 和 ArcCatalog 是 ArcGIS Server 的管理工具，可以使用这两个工具来进行服务的发布，开始停止等操作。Manager 是一个 web 应用，支持发布服务，管理 GIS Server，创建 Web 应用，以及发布 3D 服务等。ArcCatalog 包含了一个 GIS Servers 的节点，可以用来连接和管理某个 GIS Server。

2.3.6 地图内容制作工具

ArcGIS 桌面软件是 GIS 资源的编辑和制作工具，通过 ArcGIS Server 发布的各种资源都可以通过 ArcGIS 桌面软件进行制作，比如地图（mxd 文件），Geoprocessing 工具，三维场景（3dd 文件），都可以通过 ArcMap、ArcCatalog、ArcGlobe 等应用程序来进行制作。如果需要为地图服务生成缓存，可以用 ArcCatalog 来创建。

2.4 ArcGIS Server 包含的主要技术

ArcGIS Server 包含的主要技术包括 ArcSDE 技术、Web 地图应用和 ArcGIS Mobile 技术。

2.4.1 ArcSDE

企业级 GIS 是一个一体化的，多部门的系统，既要满足组织内部单一的要求，又要满足综合的需要，为 GIS 和非 GIS 人员访问地理信息和服务提供条件。数据服务器包含了要发布为服务的 GIS 资源。对于大多数 GIS 服务器，这些资源通过 ArcSDE 管理在基于关系型数据库的地理数据库（geodatabase）中。在任何一个 ArcGIS Server 的应用系统中，为了满足这种企业级需求，基于

ArcSDE 技术的长事务处理的多用户 geodatabase 都是至关重要的。因此 ESRI 将 ArcSDE 技术纳入 ArcGIS Server 体系。

ArcSDE 的优势和功能包括：

- 高效率和系统可伸缩行;
- 与 IT 系统集成;
- 发生冲突时的协调更新机制;
- 数据库复制;
- 历史归档;
- 版本和非版本编辑;
- 支持跨平台和跨数据库;
- 支持直接通过 SQL 访问 Oracle，IBM DB2 和 Informix geodatabase。

ArcGIS Server 是一个用于高级 GIS 应用的集中管理的 GIS。它可以让开发者和系统设计员实现一个集中的 GIS，支持多用户访问。集中的 GIS 应用（如 Web 应用）能够减少在每台机器上安装和管理桌面应用的费用。ArcGIS Server 的提供 Web 服务的能力，使得 GIS 能够与其他的 IT 系统有效集成，如关系数据库、Web 服务器、以及企业应用服务器。

所有级别的 ArcGIS Server 产品都包含了 ArcSDE 技术。

ArcGIS Server 工作组级含有支持 SQL Server Express 的 ArcSDE。使用这个级别的 ArcGIS Server，允许 10 个并发桌面用户和编辑人员（例如 ArcView、ArcEditor、ArcInfo、 ArcGIS Engine 应用、AutoCAD 和 MicroStation 用户）加上任意数量的服务器连接使用 SQL Server Express。SQL Server Express 是包含在 ArcGIS Server 工作组级中的一部分。它限制运行于 1CPU 或 core，最大 1GB 的内存。数据库大小最大为 4GB。管理员可以使用 ArcEditor 或 ArcInfo 来创建、管理和维护工作组级 ArcSD geodatabase。可以在 ArcCatalog 中使用 SQL Server Express 来设置和管理工作组 ArcSD geodatabase，无需额外的数据库管理知识。

ArcGIS Server 企业级包含企业级 ArcSDE 技术，这是传统的 ArcSDE 技术，它运行于 Oracle、SQL Server、IBM DB2 和 IBM Informix 数据库之上，允许任意大小的数据库、任意数量的用户，可以运行在任意配置的电脑上。使用 ArcGIS Server 企业级，用户需要自己提供 DBMS 许可。DBMS 通常由数据库管理员（DBA）管理和维护。企业级 ArcSDE 技术支持运行在跨平台上的 Oracle、IBM DB2 和 Informix，和 Window 服务器上的 SQL Server。

2.4.2　Web 地图应用

ArcGIS Server 包含一个即拿即用的 Web 地图应用，可以直接运行在 Web 浏览器中。该客户端为使用 ArcGIS Server 和其他服务提供了丰富的用户体验。这个 Web 地图应用同时也作为 ArcIMS 9.2 的一部分。Web 地图应用支持叠加多种类型的地图服务，如来自于 ArcIMS，ArcGIS Server，OGC 的 WMS 以及 ESRI 发布的 ArcWeb 服务。

Web 地图应用提供的工具有：

- 交互的内容表;
- 平滑的地图浏览，平移和缩放工具;
- 地图提示和要素查询功能;
- 空间查询和选择工具;
- 基于 Web 的 ArcSDE geodatabase 编辑功能（包括添加要素，切分，捕捉，要素修整和属性编辑）;
- ArcGIS Server 管理器提供方便的配置能力，不需要编程;
- 为.NET 和 Java 开发者提供强大的开发环境支持，提供一组可定制的编程控件和组件;
- 基于标准和开发性;
- Web 地图应用框架基于 Ajax 技术，大大增强了用户体验，它支持用户在交互使用 Web 应用的同时，应用程序与其他资源（如 Web 服务器）进行通讯。

2.4.3 ArcGIS Mobile

ArcGIS Server 为移动用户提供了名为 ArcGIS Mobile 的 Web 应用开发框架，用于创建和部属面向移动的解决方案，其特点是应用在“非实时连接”环境且面对大量用户。这些应用为运行 Microsoft Windows Mobile 的野外设备提供移动地图，GPS，无线步以及 GIS 数据复制和编辑功能。ArcGIS Mobile 支持在线和离线工作流环境中编辑版本化的 ArcSDE geodatabase。可以不用返回办公室，就可以通过 ArcGIS Server 定期进行更新同步。ArcGIS Mobile 可以运行在大量的移动设备上：智能手机、Pocket PC 和 Tablet PC。

2.5 ArcGIS Server 9.2 安装

ArcGIS Server 9.2 产品包括两个部分，一是 GIS Server，它是一个提供 GIS 服务的服务器软件产品，包括一系列核心 AO 库和一个管理这些 AO 组件的可缩放的运行环境；另一个是 ADF，即应用程序开发框架，它有 JAVA 和.NET 两种开发组件集，它是用来开发和部署基于 GIS Server 的 Web 应用程序的产品，包括组件对象、Web 控件、Web 模板和开发帮助，它还有一个 Web 程序的运行时（Runtime），专门用于发布和部署使用 ADF 开发的 Web 程序，如 ASP.NET 等。

GIS Server 是一套 GIS 服务器组件，专门用于管理和发布地图服务和定位服务，安装在 GIS 服务器上；ADF 是供开发人员使用的开发组件集，安装在开发人员的机器上，这些程序包括 Web 应用程序、Web 服务和桌面端程序，都可以使用 ADF；ADF 运行时是专门用于部署开发人员开发的 GIS Web 程序和 GIS Web Service 的工具，安装在 Web 服务器上。GIS 服务器、Web 服务器和开发人员的电脑可以是同一台机器，也可以分开安装。

2.5.1 ArcGIS Server 安装概述

ArcGIS Server 安装过程包括两个部分：安装（installation）和安装后（post installation）设置。

为了完成 ArcGIS Server 的安装，需要使用 Windows 操作系统工具手动设置一些步骤。

（1）安装过程需要用户决定安装哪些功能部件，安装程序将安装选择的部件所需要的文件。

（2）后安装是用来完成 ArcGIS Server 安装过程的。在后安装过程中，依赖于选择安装的部件，将能够配置 ArcGIS Server 和授权 ArcGIS Server。配置 ArcGIS Server 选项将设置需要的 ArcGIS Server 账号。授权 ArcGIS Server 选项将授权 ArcGIS Server 的使用。

安装 ArcGIS Server 9.2 之前必须卸载版本是 9.2 之前的以下产品：ArcGIS Desktop、ArcInfo Workstation、ArcReader standalone、ArcIMS，否则将出现冲突，不能安装。如果是 9.2 版本，则不需要卸载。

2.5.2　安装 ArcGIS Server for .NET

安装 ArcGIS Server 之前，必须确保满足以下几个条件：

- 获取 ArcGIS Server 的授权文件;
- 确认机器（和操作系统等）满足软件安装要求;
- 以具有管理员权限的账号登录操作系统;
- 确保 temp 变量被设置为一个有效的目录，并具有写的权限和可用空间;
- 确保先安装 Visual Studio 2005。

1. 安装（Installation）

在光驱中插入安装盘，点击 setup.exe，安装程序将自动开始安装。

在安装过程中，安装进程将允许选择所需要安装的部件（如图 2.4 所示）。如果需要将 GIS Server 与应用程序开发框架安装在同一台机器上，则需要同时选择 GIS Server 与 Web Application Developer Framework，如果需要分开安装，则可只选择其中一个。Web Application 用于在 Web 服务器上部署一管理 GIS Server 的 Web 应用程序。Mobile Application Developer Framework 则是用于创建移动应用程序所需要的框架。

在安装 ArcGIS Server 时，其安装目录取决于第一次安装 ArcGIS 产品的目录。比如第一次在 D 盘安装了 ArcGIS Desktop，那么下次安装 ArcGIS 产品时将默认安装在 D 盘。因此，必须保证安装目录有足够的空间。

根据提示，点击 Next，完成安装。当完成了 ArcGIS Server 安装后，将进行后安装。

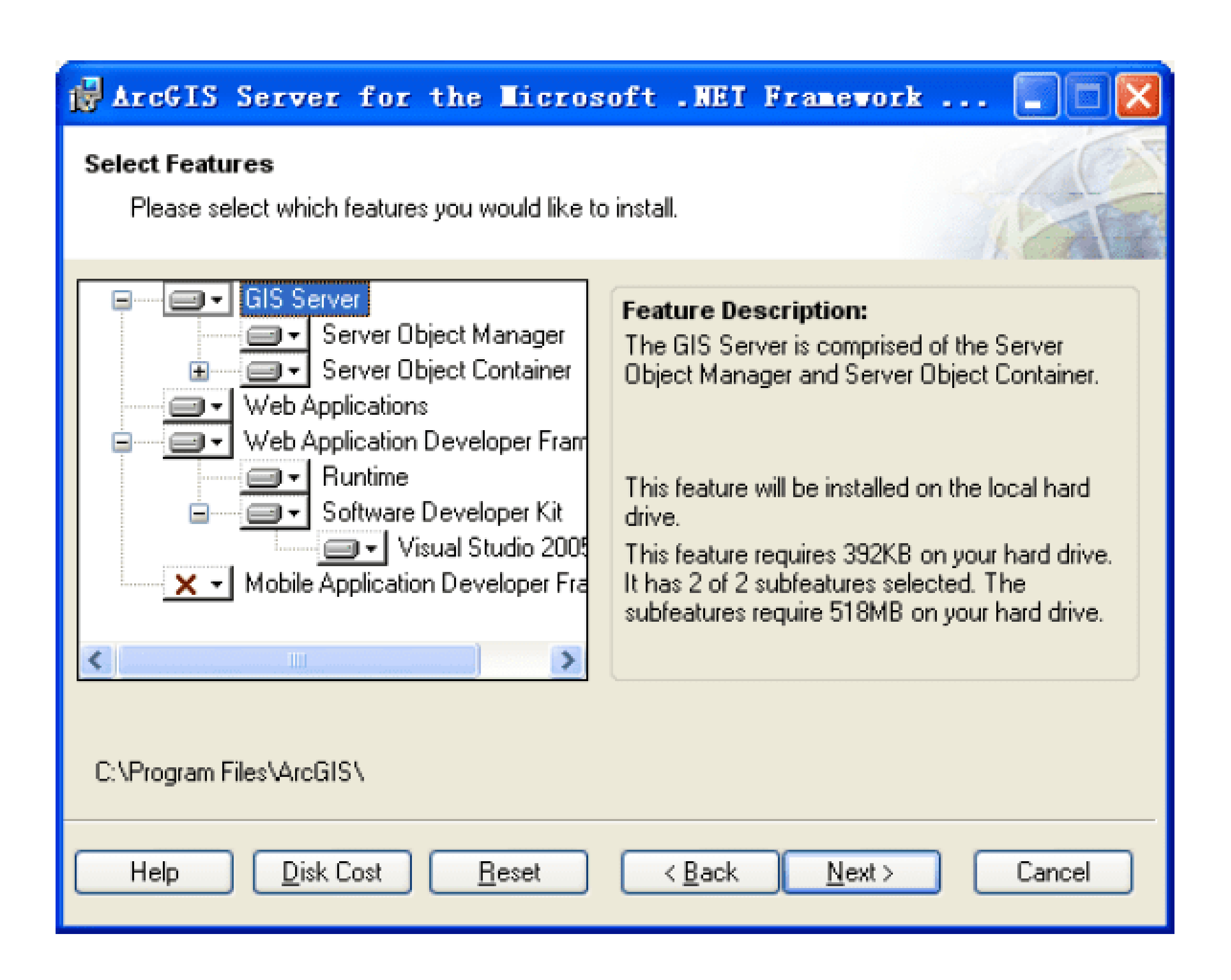


图 2.4　ArcGIS Server 安装选项

2. 后安装（Post Installation）

使用 ArcGIS Server 必须完成安装后设置。可以在初始安装时完成安装后设置或是在稍后运行安装后设置（开始程序 | ArcGIS | ArcGIS Server | ArcGIS Server Post Installation）。

（1）进入 GIS Server Post Install 对话框，在对话框中有两个安装选项，分别是配置 ArcGIS Server 和认证 ArcGIS Server，如图 2.5 所示。如果是安装服务器对象容器（SOC），则两个选项都必须被包括。

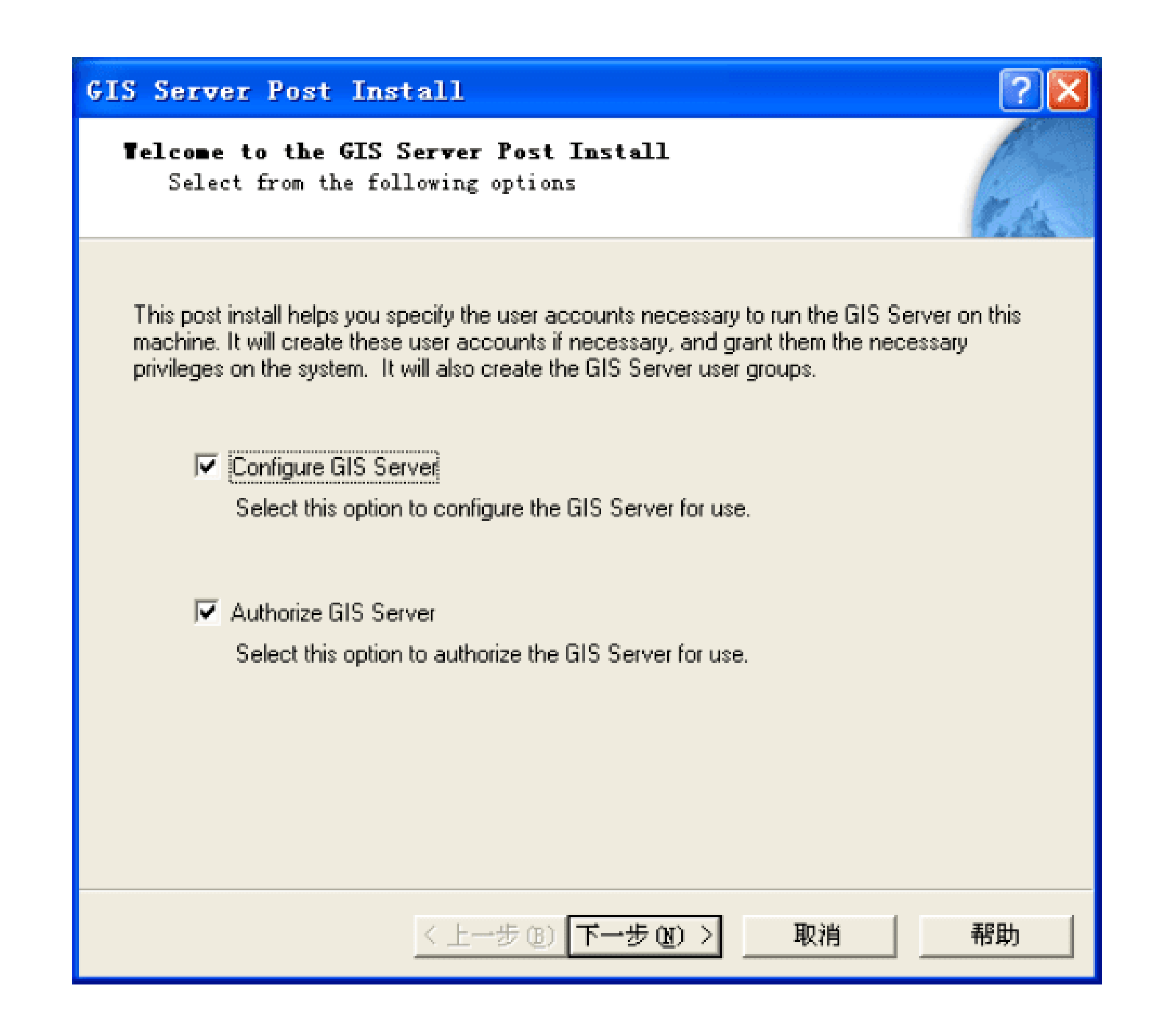


图 2.5　后安装选项

（2）选择“下一步”，进入配置 ArcGIS Server 选项对话框，如图 2.6 所示。

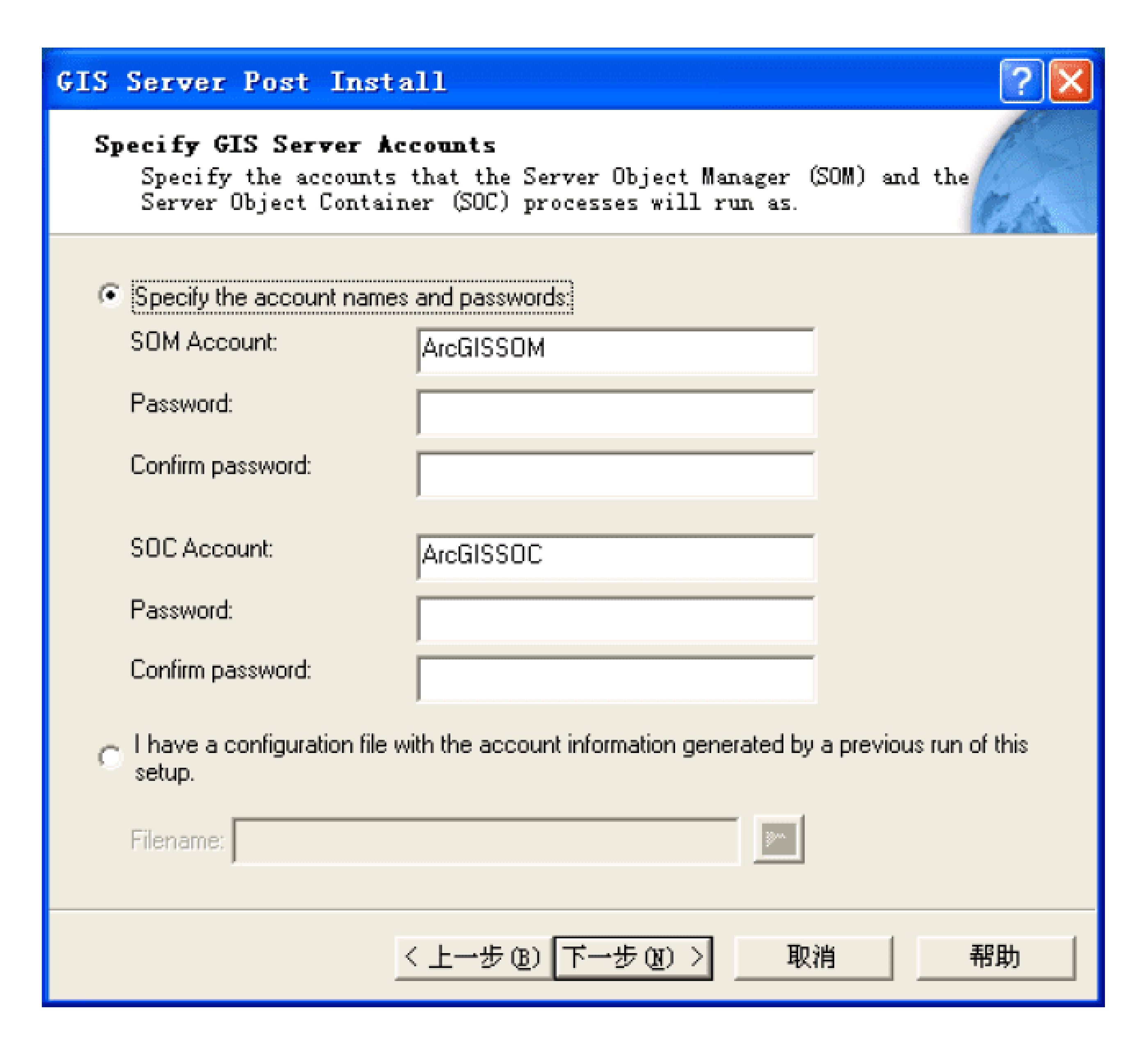


图 2.6　指定 GIS Server 账号与密码

（3）配置 ArcGIS Server 选项将设置 ArcGIS Server 账号。指定 ArcGIS Server 账号和密码，以及 ArcGIS Container 账号和密码。ArcGIS Server 账号运行服务器对象管理器（SOM）服务和进程。这个进程管理容器机器上的容器进程，并且管理 ArcGIS Server 的配置日志文件。ArcGIS 容器账号是容器进程管理服务器对象的账号。容器进程是由服务器对象管理器（SOM）启动的，但是是作为 ArcGIS 容器账号运行的。ArcGIS 服务器账号和 ArcGIS 容器账号能够是本地账号或是域账号。推荐用本地账号。如果本地账号不存在，安装后设置将会创建它。如果指定域账号，这些账号必须是已经存在的。

（4）设置 ArcGIS SOM 用户和 ArcGIS SOC 用户的密码后，选择“下一步”，进入“设置 Webservices 账号”对话框。Webservices 账号用于 Web 服务器连接 GIS 服务器，以处理 Web 服务请求。设置好账号与密码后，选择“下一步”，进入“指定 GIS Server 目录”对话框。保持默认选项，进入代理服务器的设置。同样保持默认选项，即不使用代理服务器。选择“下一步”，进入“输出服务器配置文件选项”对话框，默认选择，进入下一个对话框，选择 Install 按钮来安装。

（5）当配置 ArcGIS Server 以后，后安装程序进入认证向导。认证向导的第一个对话框是注册选项，选择其中的第三个，即已有认证文件，进入下一步。输入认证文件所在路径后，选择“下一步”，对话框中将列出被授权的功能。

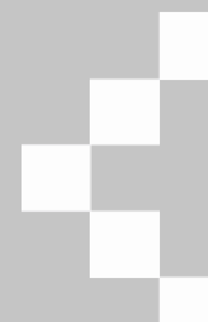
至此安装完成，但是配置任务并没有结束，还需要配置组账户。

GIS Server 安装后，在计算机上创建了两个本地组账户，分别是 agsadmin 与 agsusers。agsadmin 组中的用户将可以拥有管理 GIS Server 的权限，agsusers 组中的用户将拥有使用 GIS Server 的权限。

将自己最常用的账户（通常是 Administrator）加入到这两个组中。设置完以后，需要重新启动计算机或注销用户一次。

第 3 章

ArcGIS Server管理与服务发布



一个 GIS 服务器可发布多种类型的 GIS 服务。一个 GIS 服务代表一个 GIS 资源，例如二维地图、三维地图、地理编码或空间数据库连接等。客户端应用程序在服务器上查找这些服务后，便可访问这些服务。通过服务的方式简化了在不同客户端共同享用资源，例如可以确保每个客户端显示的资源是一致的，而且由于只需要在服务器上存储资源，不需要在客户端安装 GIS 软件，因此可节约费用。一切的功能都在服务器上完成，包括存储资源，发布服务，进行 GIS 运算，将结果以常用的格式（图片或文本）发送给客户端。

ArcGIS Server 可发布多种服务，包括地图服务、地理编码服务、网络分析服务、空间处理服务等，这些服务的发布与管理可以通过 Manager 应用或 ArcCatalog 应用来完成。然而在发布这些服务之前，需要利用桌面产品进行创建和准备这些服务使用的资源，例如 mxd 文件、3dd 文件等。

本章将介绍：

3.1　管理 ArcGIS Server
3.2　发布服务
3.3　配置文件的使用

3.1　管理 ArcGIS Server

用户可以登录到 Manager 或者是在 ArcCatalog 中创建管理的连接来对 GIS Server 进行管理，首先一个必须要做的是添加相应的 ArcSOC 的机器。在用户的整个系统配置中，肯定有一台或者多台的 ArcSOC 机器，所以需要在 Server 属性中进行添加，如果把 ArcGIS Server 只是安装在一台机器上，那么就把本机作为 ArcSOC 的机器即可。

3.1.1　使用 Manager 管理 ArcGIS Server

要使用 Manager 来管理 ArcGIS Server，需要在安装 ArcGIS Server 时选择安装此工具。然后需要将一个账户（我们这里以 Administrator 为例）加入到 agsadmin 组中。

Manager 工具是一 Web 应用程序。这就意味着可以从其他计算机通过浏览器远程管理 ArcGIS Server。

1．登录 Manager

从安装了 ArcGIS Server 计算机上选择“开始 | 程序 | ArcGIS | ArcGIS Server for the Microsoft .NET Framework | ArcGIS Server Manager”，或从其他计算机的浏览器中输入如下格式的地址 http://192.168.2.245/ArcGIS/Manager/login.aspx（其中 192.168.2.245 是笔者试验用的 GIS 服务器的 IP 地址，请换成读者自己计算机的 IP 地址），将打开如图 3.1 所示的登录界面。

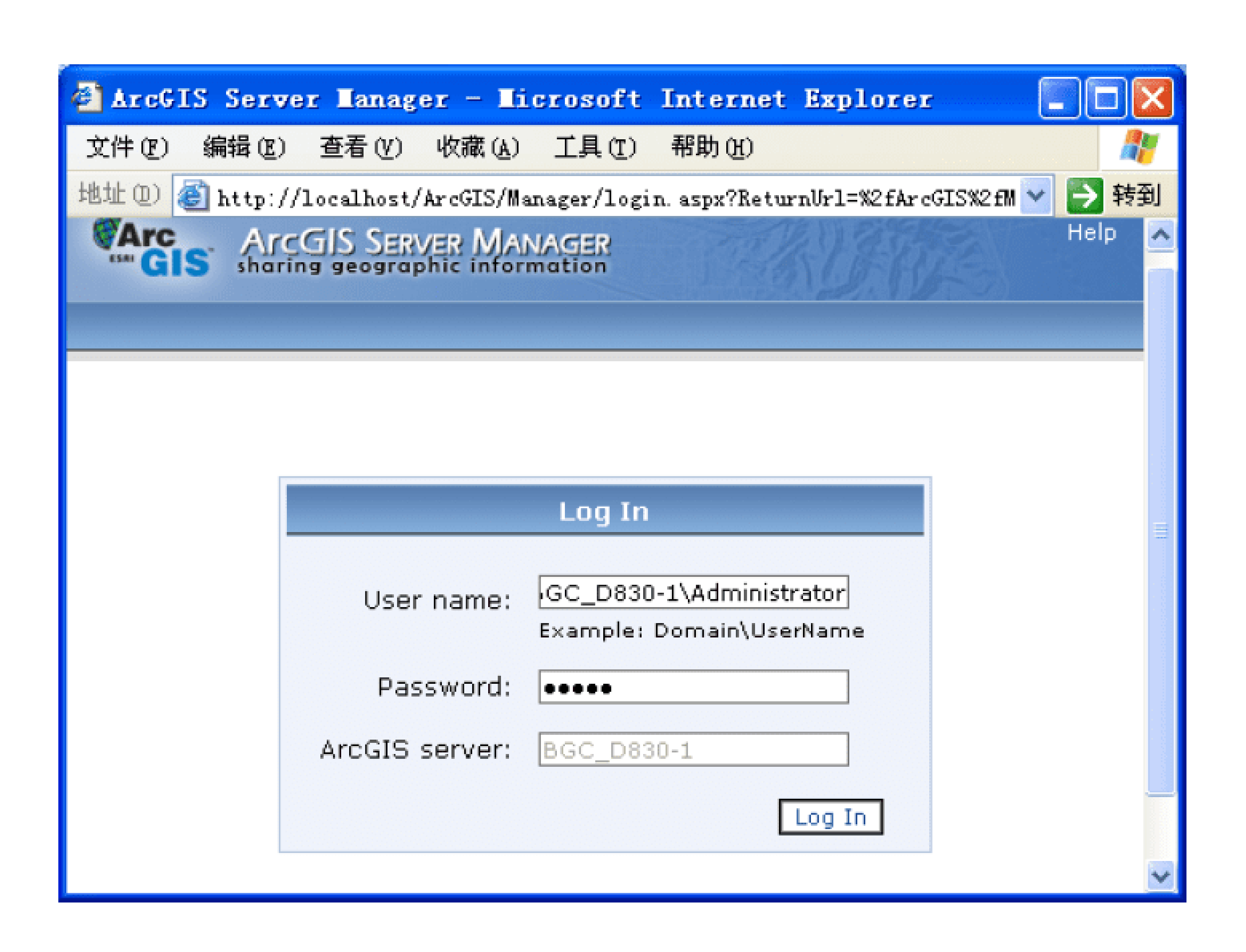


图 3.1　ArcGIS Server Manager 登录界面

在该登录界面的用户名中需要输入“域名\用户名”，如果没有使用域，那么域名就是 GIS 服务的计算机名，也就是该界面中第三个文本框中的名称。而这个名称在登录界面中是不能更改的，这是因为 Manager 工具直接与 ArcGIS Server 的一个实例相关。只能用 Manager 工具登录一个 GIS

服务器。

2. 服务对象容器管理

当第一次连接 GIS 服务器后，需要在服务器上增加一个或多个服务对象容器，或 SOC 计算机。SOC 可以装在多台机器上，是服务对象的宿主。每一台 SOC 机器可以产生多个容器进程，一个容器进程可以运行多个服务对象。为了让 GIS 服务器能够利用这些 SOC 计算机，需要利用服务对象管理器 SOM 连接它们。

要让一计算机作为 SOC，需要在安装 ArcGIS Server 时选择安装“Server Object Container”。此外，还需要在后安装时指定与 ArcGIS Server 相同的 SOM 与 SOC 账号与密码。

当然如果在一服务器上既安装了 SOM 又安装了 SOC，而没有其他 SOC 计算机，那么就没有必要添加 SOC。

要添加服务对象容器计算机，需要在 Manager 工具中选择 GIS Server 标签，然后选择 Add Host Machine，输入计算机名称后，便可添加。

3. 服务器路径管理

服务器路径就是网络上的一个物理路径，该路径可以被所有容器计算机访问。当需要时，GIS 服务器在这些路径中保存临时文件。服务器会根据用户指定的间隔时间定期清除服务器路径中的文件。

服务器路径可分三类，分别是缓存、任务与输出。

- ❑ 缓存路径是为了加快显示速度，存储了地图服务的预描绘地图块。要创建地图缓存，必须使用 ArcCatalog，Manager 工具不能创建。
- ❑ 任务路径存储了空间处理服务需要的文件。通常，空间处理服务需要一定的空间来保存临时文件与存储后续工作的信息。
- ❑ 输出路径通常用于服务器保存临时文件。空间处理服务必须有输出路径，空间数据服务推荐使用输出路径，对于地图服务，输出路径则是一可选项，而对于其他服务，则不需要输出路径。

在 ArcGIS Server 的后安装过程中，在指定的位置默认为每一个类型创建了一路径。默认路径为 c:\arcgisserver。

可以通过 Manager 工具的 GIS Server 标签中的 Directories 来管理服务器路径，包括新加、删除与编辑。

4. 日志文件位置

ArcGIS Server 将系统消息记录在日志文件中。当怀疑服务器工作不对时，可通过检查日志文件来查看服务器运行情况。

默认的日志文件在 ArcGIS Server 安装目录中的 Server\user\log 路径中，当然可以通过 Manager 来指定日志文件的路径。

如果 ArcGIS Server 配置的是分布式安装，即服务对象运行在多台计算机上，那么需要共享该日志路径，并使用一 UNC 路径（例如\\myServer\log）引用。还需要确保 SOM 与 SOC 账号对该路

径有读写权限。

5. 设置服务对象容器的能力

通过设置服务对象容器的能力属性，可控制每一个服务对象容器计算机可运行多少个服务实例。如果一台服务对象容器计算机的性能大大高于其他计算机，这时可将该计算机的功能值设置大些。

要设置服务对象容器的能力，在 Manager 工具中选择 GIS Server 标签，在左边的菜单中选择 Host Machines（SOC），然后在需要设置能力的计算机旁选择 Edit 连接，然后在 Capacity 文本框中输入一理想的数值。默认值为-1，表示无限制能力。最后选择 Update 来保存设置。

3.1.2 使用 ArcCatalog 管理 ArcGIS Server

可以将 ArcCatalog 想象为 ArcGIS Server 的一个用户界面。通过 ArcCatalog，可以查看或管理（如果是管理员）服务器上运行的服务的设置。ArcCatalog 为 GIS 服务器提供两个不同的界面，一个供管理员用，另一个供普通使用者使用。

当作为普通用户连接 GIS 服务器后，ArcCatalog 只简单地列出该用户可使用的服务。该用户可使用这些服务，例如在 ArcMap 中显示服务器中的地图，但是不能管理这些服务，例如删除。当作为管理员连接时，可看到一些额外的工具允许管理服务。这时可以通过 ArcCatalog 管理运行在服务器上的服务以及该服务器所包含的一组计算机，可以监视客户端应用程序如何使用每一个单独的服务，以及是否有足够的资源满足客户端需求。可能常常需要为某一特定的服务增加计算机资源，有时还可能需要增加额外的计算机来分担负载。

1. 连接 GIS 服务器

ArcCatalog 提供了两种连接 GIS 服务器的选择，一是管理员连接，另一是普通用户连接。

（1）管理员连接

启动 ArcCatalog 后，在目录树中双击 GIS Servers，然后双击 Add ArcGIS Server，在随后的 Add ArcGIS Server 向导的第一个页面中选择 Manage GIS Services，然后进入 General 页面，如图 3.2 所示。

在该页面的 Server URL 中，输入需要连接的 ArcGIS Server 实例的 URL。该 URL 的格式为“http://<服务器名称或 IP 地址>/<实例名>/services”，实例名是在后安装过程中指定的，默认为 arcgis，例如连接笔者的 GIS 服务器的 URL 为 http:// BGC_D830-1/arcgis/services 或为 http://192.168.2.245/arcgis/services。对于 Host Name，需要输入的是服务器的计算机名称。如果 GIS 服务器包含了几台计算机，那么这里需要指定的是运行服务对象管理器的计算机。输入完成后，选择 Finish，即可在目录树中见到服务器。

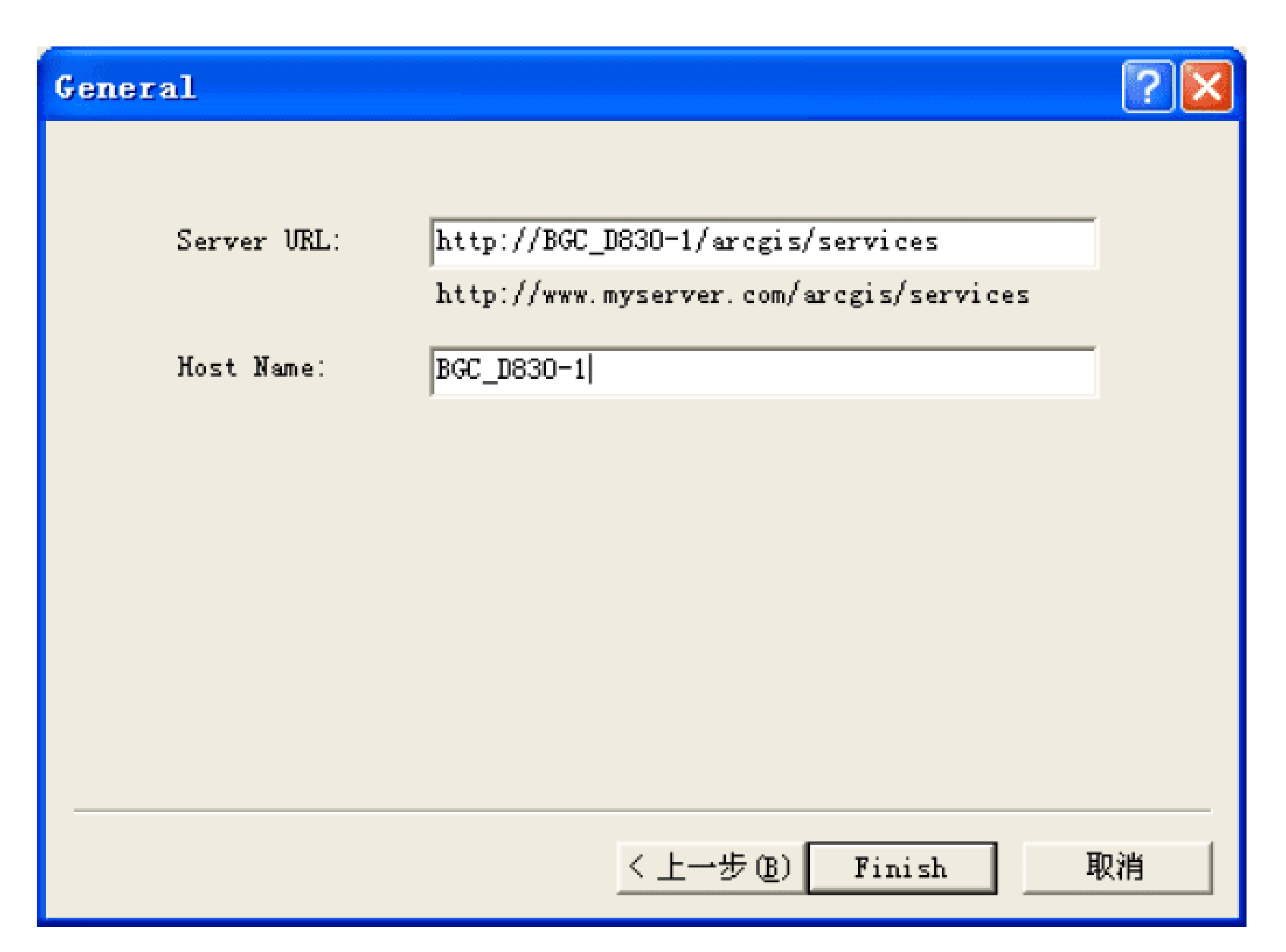


图 3.2 管理员连接界面

（2）普通用户连接

以普通用户连接时，只能查看与使用服务器上的服务，不能编辑服务的属性，也不能增加、删除、启动、停止或暂停服务。当使用普通用户连接时，可选择是连接到局域网中的本地服务器，还是广域网中的远程服务器。当使用局域网连接时，必须使用服务器中 agsusers 用户组中的用户运行 ArcCatalog。

需要以某个用户的身份来运行 ArcCatalog 时，可以通过“属性”中的“快捷方式”选项卡中“高级”按钮，设置为“以其他用户身份运行”。也可以用命令行启动 ArcCatalog，到 ArcGIS 安装目录下输入：runas /user:*** arccatalog，而不用切换用户。

以普通用户身份连接的步骤如下：

（1）启动 ArcCatalog 后，在目录树中双击 GIS Servers，然后双击 Add ArcGIS Server，在随后的 Add ArcGIS Server 向导的第一个页面中选择 Use GIS Services，然后进入 General 页面，如图 3.3 所示。

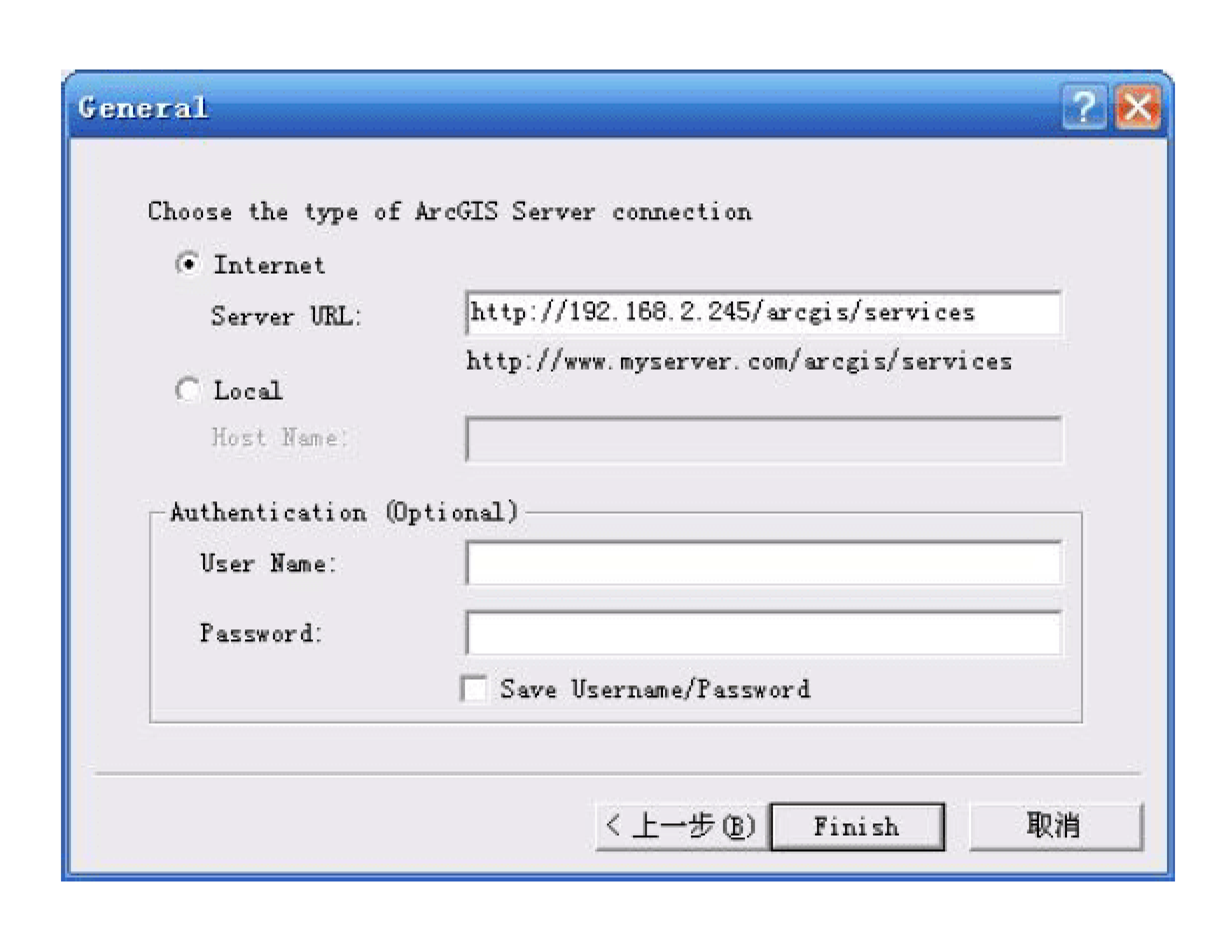


图 3.3 以普通用户身份连接界面

（2）首先选择连接方式。如果要通过广域网连接，选择 Internet，并输入服务器的 URL，该 URL 的格式为“http://<服务器名称或 IP 地址>/<实例名>/services”，实例名是在后安装过程中指定的，默认为 arcgis，例如连接笔者的 GIS 服务器的 URL 为 http:// BGC_D830-1 /arcgis/services 或

为 http://192.168.2.245/arcgis/services。

（3）如果服务器不允许匿名连接，那么还需要输入用户名与口令。如果要连接的是局域网中的服务器，则选择 Local，并输入服务器的计算机名称即可。

2．管理服务对象容器

使用 ArcCatalog 管理服务对象容器的方法如下：

以管理员连接方式连接服务器，然后在 ArcCatalog 的目录树中，右击 GIS 服务器名称，选择上下文菜单中的 Server Properties 项。在弹出的“ArcGIS 服务器属性”对话框中选择 Hosts 标签，进入该标签页。在该标签页中选择 Add 按钮，弹出 Add Machine 对话框，如图 3.4 所示。输入服务对象容器计算机的名称，或利用浏览按钮选择一台计算机。

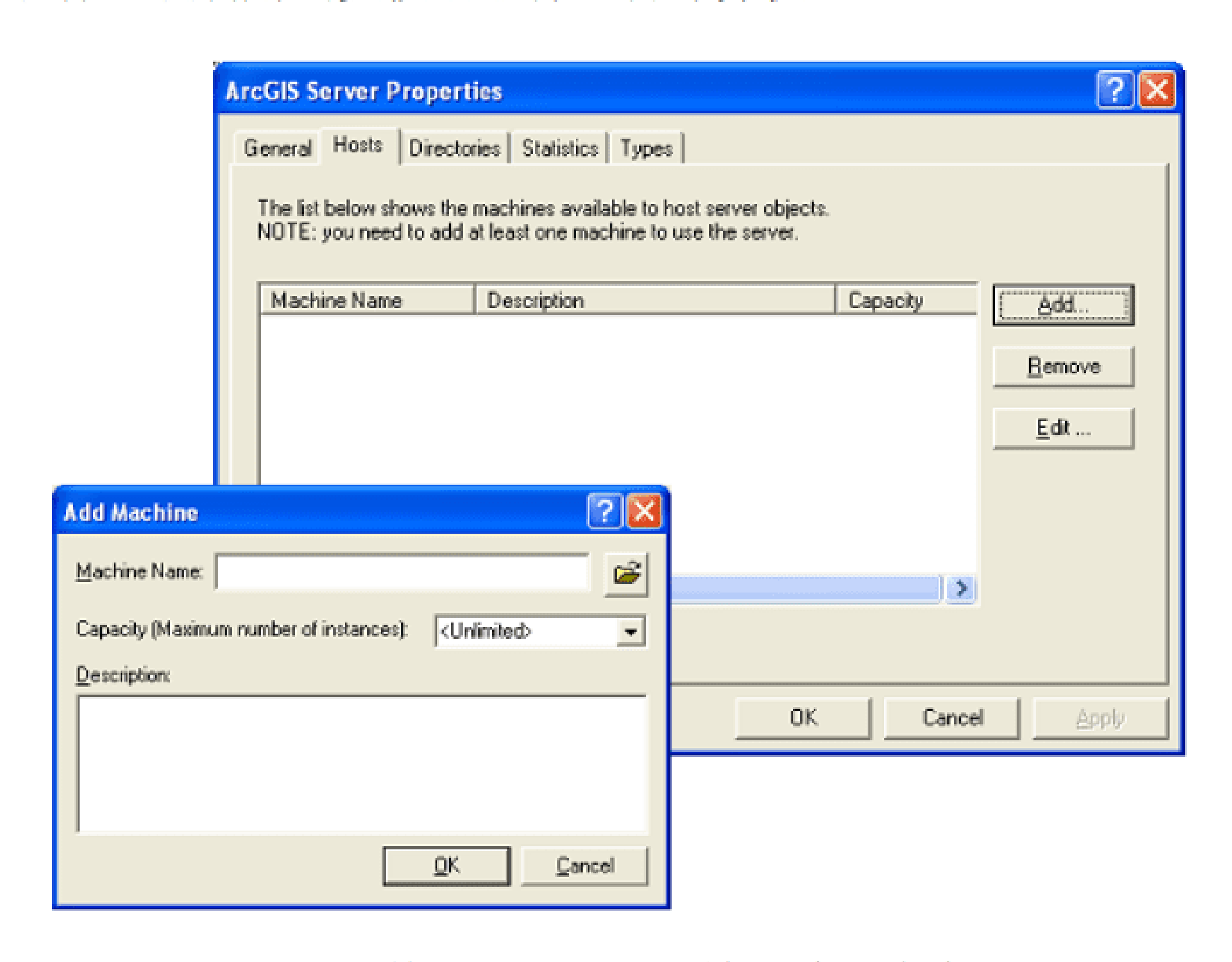


图 3.4　利用 ArcCatalog 增加服务对象容器

利用 ArcCatalog 的 ArcGIS Server Properties 对话框也能管理服务器路径、日志文件以及服务对象容器能力，操作方法很简单，这里为节省篇幅，不再赘述。

3.2　发布服务

在 ArcGIS Server 中发布的资源是通过桌面产品进行创建和准备的，比如 mxd 文件，比如 3dd 文件等，因此在发布服务之前要准备好这些数据，并使得 agsusers 组中的用户可以有权限访问到这些数据。发布服务的功能在 Manager 应用以及 Arccatalog 中都可以完成。

3.2.1　服务与功能

当在 GIS 服务器上将一个 GIS 资源发布为服务时，可以指定该服务包含的功能。功能即客户使用该服务的方法，例如地图服务最基本的功能就是绘制地图，再例如可以让一服务提供一地理定

位功能，以便客户能通过地址定位地图。

作为 ArcGIS 服务器管理员，主要看重的是 GIS 资源以及服务。而对于用户，则更关注服务器发布的服务所能提供的功能。因此，作为管理员在将 GIS 资源发布服务时，应使该服务包含不同功能；作为服务的用户，则可以将这些功能看成不同的服务。

但是不是所有类型的资源都可以包含所有类型的功能。一个资源所能提供的功能取决于该资源的类型。表 3-1 列出了可包含某种功能的资源类型。如果是地图文档，那么还取决于里面的图层。地图文档可包含功能最多，包括 WMS 与 KML。其他资源，例如空间数据访问与空间处理，需要特殊类型的图层。

表 3-1 资源类型与功能

功能	功能描述	所需 GIS 资源
地图绘制	提供访问一地图文档内容。当发布一地图服务时，默认包含该功能	地图文档（.mxd）或发布后的地图文档（.pmf）
WMS	使用地图文档创建 OGC 的 WMS 规范要求的服务	地图文档
移动数据访问	允许从地图文档中提取数据到移动设备中	地图文档
KML	使用地图文档创建锁孔标记语言（Keyhole Markup Language）要素	地图文档
空间数据访问	允许用户通过 ArcMap 执行复制与数据提取。要提供该功能，需要在发布地图文档时选择创建相应的空间数据服务	包含来自空间数据库中图层的地图文档
空间处理	提供访问空间处理模型。工具图层代表从 ArcToolbox 中拖动到地图文档内容列表中的模型。当发布地图文档时，可选择创建该功能。当发布空间处理服务时，始终包含该功能	Toolbox 文件（.tbx）或包含工具图层的地图文档
网络分析	使用网络分析扩展解决网络分析问题	包含网络分析图层的地图文档
三维地图	提供访问三维地图文档中的内容。当发布三维地图服务时，始终包含该功能	三维地图文档（.3dd）
空间数据	提供空间数据库数据查询、提取与复制功能。当发布空间数据服务时，始终包含该功能	SDE 连接文件（.sde）、个人空间数据库、文件空间数据库、或包含来自空间数据库中图层的地图文档
地理编码	提供地址定位功能。当发布地理编码服务时，始终包含该功能	地址定位器文件（.loc）、ArcView 3 定位器、ArcSDE 定位器、个人空间数据库定位器、文件空间数据库定位器

地图文档发布方法：

在 ArcMap 中发布地图，需要先勾选菜单 Tools-Extensions 中的 Publisher，载入该模块。打开 Publisher 工具栏，点击 Publish Maps 按钮，导出一个 pmf 文件。这个文件可以在 ArcReader 中打开，但是还不能分发，因为它指向的是硬编码的本地磁盘中的路径。接下来点击另一个 Create Data

Package 按钮，选择刚才的 pmf 文件。接下来将弹出对话框，可以选择存放发布的地图的目录以及数据文件的形式，还有扩展选项（导出全部要素或栅格还是只导出当前视图的部分）。完成后的目录中出现了 data 和 pmf 两个子目录，前者是编码后的数据，后者即发布的地图。

同样的操作可以在 ArcCatalog*中完成，勾选 Publisher 扩展后，可以右键点击 pmf 文件，选择 Create Data Package。

KML

KML 全称是 Keyhole Markup Language，是一个基于 XML 语法和文件格式的文件，用来描述和保存地理信息如点、线、图片、折线并在 Google Earth 客户端之中显示。

3.2.2　发布与管理服务

使用 Manager 或 ArcCatalog 都可以发布与管理服务。

1. 使用 Manager 发布与管理服务

以发布与管理地图服务为例来说明。

按照 3.1.1.节中介绍的方法，使用 agsadmin 组中的账户登录到 Manager。切换到 Services 选项卡中，点击 Publish a GIS Resource 链接，进入发布 GIS 资源向导的第一个页面，如图 3.5 所示。在该页面中，首先需要在 Resource 中输入资源的文件名（包含绝对路径），例如安装程序自身带的北美地图文档 NorthAmerica.mxd，默认在 C:\Program Files\ArcGIS\DeveloperKit\SamplesNET\Server\data\NorthAmerica\mxd 目录中。为该地图服务输入一个名称，例如 NorthAmericaMap。

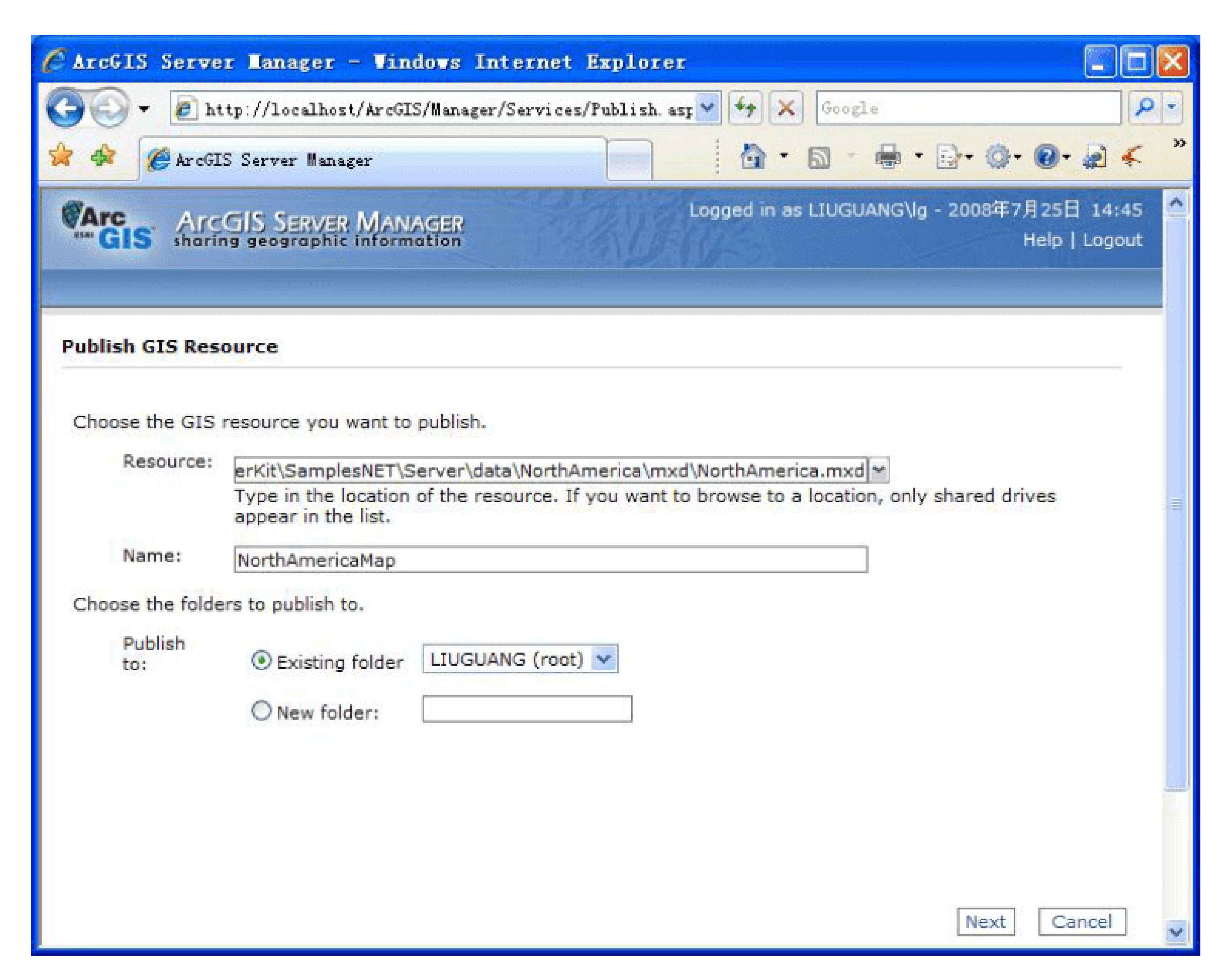


图 3.5　发布 GIS 资源向导——指定资源与服务名称

输入资源与名称参数后，选择 Next 按钮进入下一个页面，在该页面中，需要设置服务可提供

Web GIS

的功能。如果发布的是一个地图文档资源，那么地图功能为默认选项，此外还可选择 WMS、Mobile Data Access 与 KML。如果地图文档资源中的数据来自数据库，那么还可选择 Geodata Access。我们这里使用默认值，直接选择 Next 按钮进入最后一个页面。该页面进行了总结，告知用户将创建 NorthAmericaMap.MapServer GIS 服务，同时将创建一个 Web 服务，地址为 http://liuguang/arcgis /services/NorthAmericaMap/MapServer。选择 Finish 按钮完成发布服务，并返回到 Services 页面。

这时 NorthAmericaMap 服务将列在 Services 页面中的服务列表中，单击该服务左边的加号按钮，如果能正确显示图形，如图 3.6 所示，表明服务发布成功。

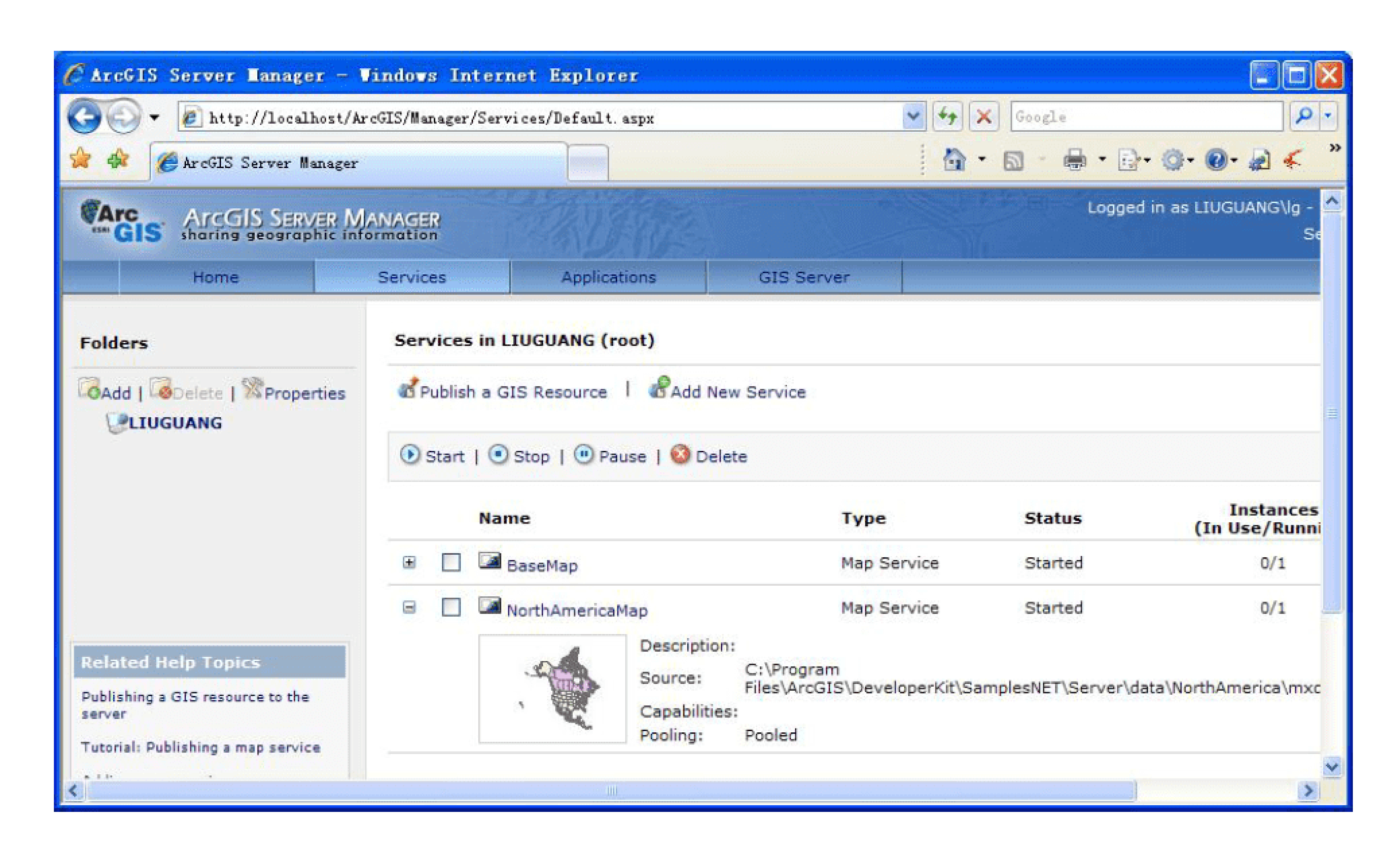


图 3.6　正确发布地图服务后可显示地图

利用 Services 页面上的 Start、Stop、Pause 及 Delete 按钮，可启动、停止、暂停与删除选中的服务。

要设置某个服务的参数，例如需要增加某些功能，可在 Services 页面中选择该服务的 Edit 按钮，进入服务编辑页面。通过服务编辑页面左边的 General、Parameters、Capabilities、Pooling 等，可设置服务的启动类型、图片输出方式、功能以及实例个数等。

2. 使用 ArcCatalog 发布与管理服务

要使用 ArcCatalog 发布与管理服务，需要以管理员方式连接服务器。然后选择右键菜单的 Add GIS Service 命令，进入增加 GIS 服务向导。在第一个页面中输入要发布的服务的名称，例如 NorthAmericaMap。然后进入下一个页面，在这里输入地图文档的文件名（包含绝对路径）。然后进入功能设置页面，选择需要的功能后，进入是否池化界面。一般选择池化，然后根据需要设置服务实例的个数。其他页面中的值一般使用默认值即可。

在 ArcCatalog 中发布服务的另一个途径是，通过目录树找到某资源文件，然后选择右键菜单的 Publishi to ArcGIS Server 命令，即可发布服务。操作过程与使用 Manager 发布服务类似，这里不再赘述。

服务的管理可以使用右键菜单的 Service Properties 命令打开属性窗口来完成。

3.2.3 配置地图缓存

1. 为什么要使用地图缓存？

ArcGIS Server 在发布地图时可以使用缓存来显著提升性能。

地图缓存是一个包含了不同比例尺下整个地图范围的地图切片的目录。如图 3.7 所示，该地图中包含了两个比例尺的图片缓存，这样在服务器响应客户端的地图请求时，不需要动态生成地图图片，而是将这些切片返回给客户端。从缓存中返回一个切片远比动态生成地图要快得多。ArcGIS 的桌面应用程序以及 ArcGIS Server Web ADF 程序可以使用通过虚拟目录来在地图服务中使用切片缓存。

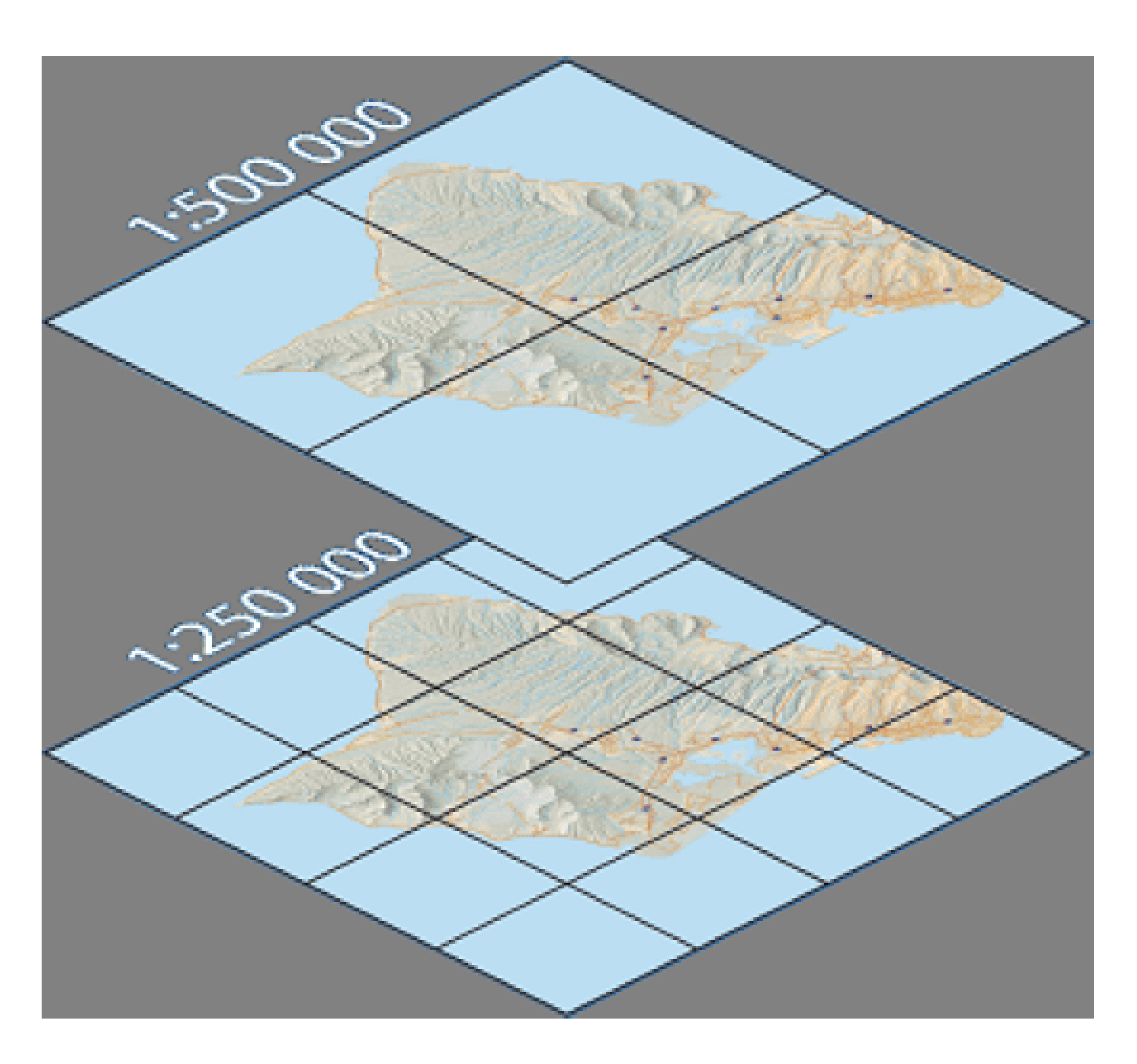


图 3.7　地图切片示意图

即一个缓存的地图服务就是能够利用静态图片来快速提供地图的服务。一个完整的缓存地图服务使用以下几个组件来完成其工作：

- 缓存。这是由包含了不同等级集合的缓存地图图片以及描述其缓存的结构的文件（conf.xml）；
- Web 服务器。Web 服务器引用实际的地图服务，一个虚拟的地图缓存目录与真实的缓存目录相对应，同时使用切片处理服务，而其虚拟路径是不能直接访问的；
- ArcGIS 服务器。ArcGIS 服务器上运行着提供地图缓存信息的地图服务，支持查询与数据操作，当缓存不存在或不可用时，生成地图切片。

有两种不同类型的地图缓存服务，分别是单个融合缓存与多图层缓存。

在单个融合缓存模式下，在每个比例尺下所有图层融合在一块，创建地图切片。这时融合的缓存显示为一整体图层，不允许单独设置某个图层的可见性，以及选择要素与调整注记。多图层缓存在某个比例尺下为每个图层单独创建地图切片，这时客户端看到的是图层集合，可以控制每个图层的可见性、注记以及要素选择。

当然，融合缓存的性能高于多图层缓存。利用融合缓存建立的应用程序不使用父地图服务，而是直接利用缓存路径中图片或使用切片处理服务。

地图服务缓存路径位于服务器的缓存路径中，默认为 C:\arcgisserver\arcgiscache。某个具体的地图服务缓存在该路径中创建一与地图服务同名的路径。在该路径下是一子路径，名称为地图中一数据框架的名称。该路径中的内容因为创建不同类型的缓存而不同。如果创建的是融合缓存，那么是一个名为_alllayers 的子路径；如果创建的是多图层缓存，那么是每个图层的子路径。在每个图层或_alllayers 的子路径中，是每个详细级别的子路径，这些详细级别对应不同的比例尺。在每个级别的子路径中是每切片行的子路径，在该路径中存储的是实际的图片文件，代表该行中某具体列的地图。图 3.8 是使用融合缓存的地图服务缓存目录结构。

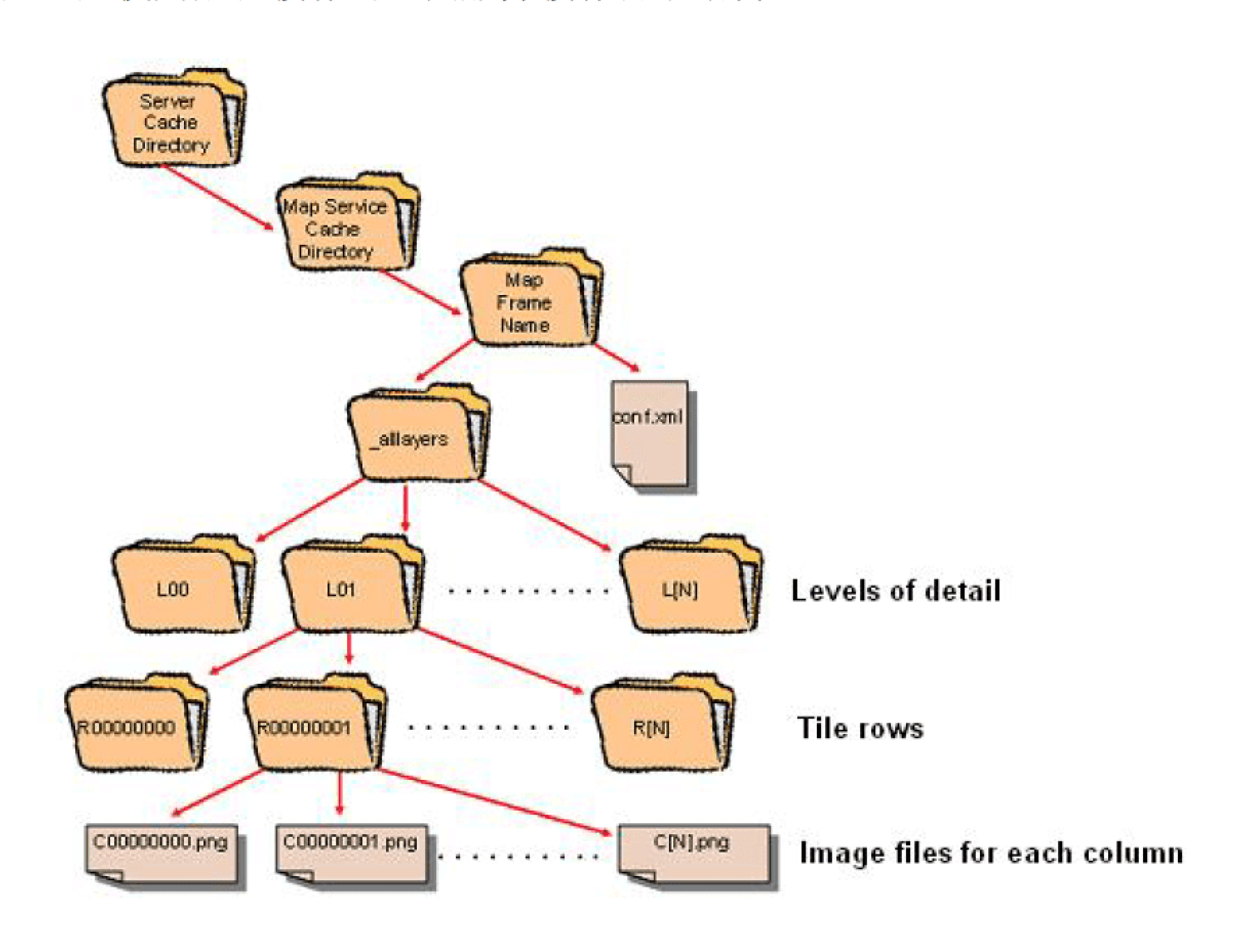


图 3.8 融合缓存类型地图服务缓存目录结构

每个地图缓存服务缓存路径中都有一个名为 conf.xml 的文件（见图 3.8），该文件完整描述了缓存，包括其组织结构以及地图文档的空间引用与切片网格的对应关系。切片网格使用详细级别、行与列来描述。例如，在 L00 详细级别中 R00000000 行的名为 C00000000.png 的地图切片对应最小比例尺级别中最左上角的切片网格。

地图切片的像素尺寸用户可选择，可选项包括 128、512 或 1024 像素，图片格式可为 PNG8、PNG24、PNG32 或 JPEG。图片根据默认的透明度设置背景颜色。如果地图文件中没有设置背景颜色，那么默认为 253，253，253。该值可使用 ArcMap 来修改。

使用缓存地图服务最大的好处是可以动态地改进客户端用来显示复杂的地图所花费的时间。一个客户端使用缓存地图服务获取和显示地图时仅仅只是受到带宽的限制，由于动态地图服务使用阴影和其他高级的图像属性渲染和显示地图，因此地图质量比静态地图服务要好，但是所花费的时间也多，如图 3.9 所示。但是，如果使用了缓存地图服务，由于地图切片是事先生成的，而不需要动态绘制，因此既可以保证地图图像质量，又可以快速响应客户端的地图请求。

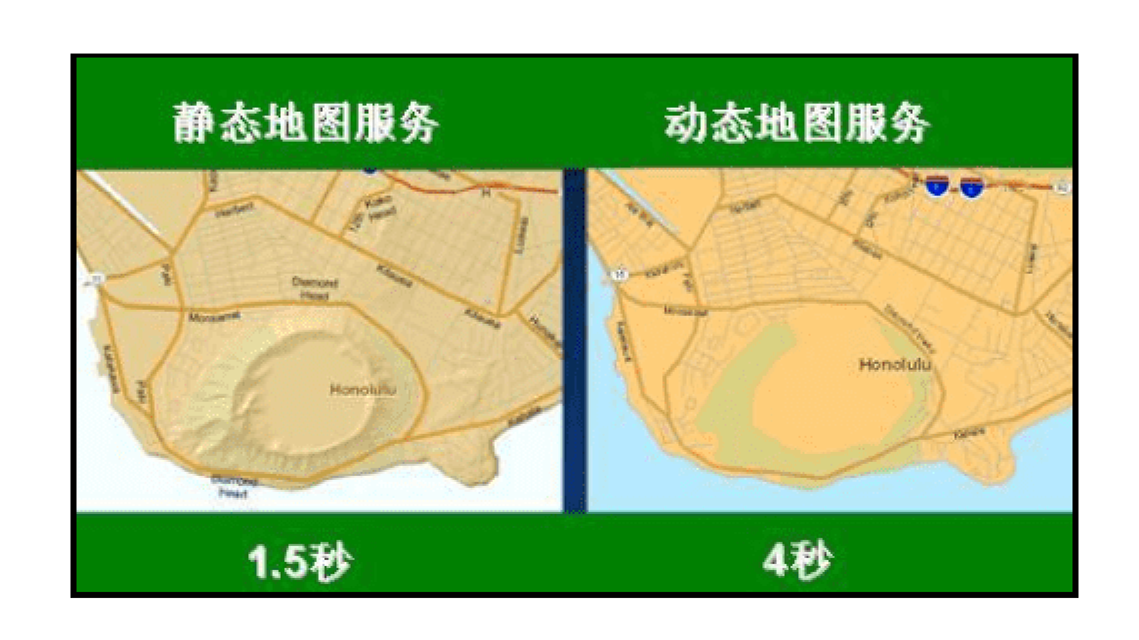


图 3.9　静态地图服务与动态地图服务在图像质量与效率上的对比

缓存地图服务使用的服务器资源远少于动态地图服务。客户端与服务器初始连接后，一旦发现有缓存结构的存在，就不再需要与服务器打交道，而是直接从缓存路径中得到地图切片。

2. 创建地图缓存

ArcGIS 提供了多种方式来创建一个缓存地图服务，ArcCatalog、ArcToolBox、ArcToolBox、脚本语言命令行或者 ArcEngine API 都可以创造缓存地图服务，相对而言，ArcCatalog 提供的方式简易易懂。

启动 ArcCatalog，进入 GIS 服务器管理界面，选择需要建立缓存的服务后右击选择 Service Properties，在其服务属性中选择 Caching 选项卡。在其中单击 Generate 按钮进入 Generate Map Server Cache 对话框，如图 3.10 所示。在该对话框中需要对生成地图服务缓存进行参数设置，主要设置的是参数比例尺的级别数，其他参数都保持默认值即可。填写完成后，单击 OK 就可以生成地图缓存。

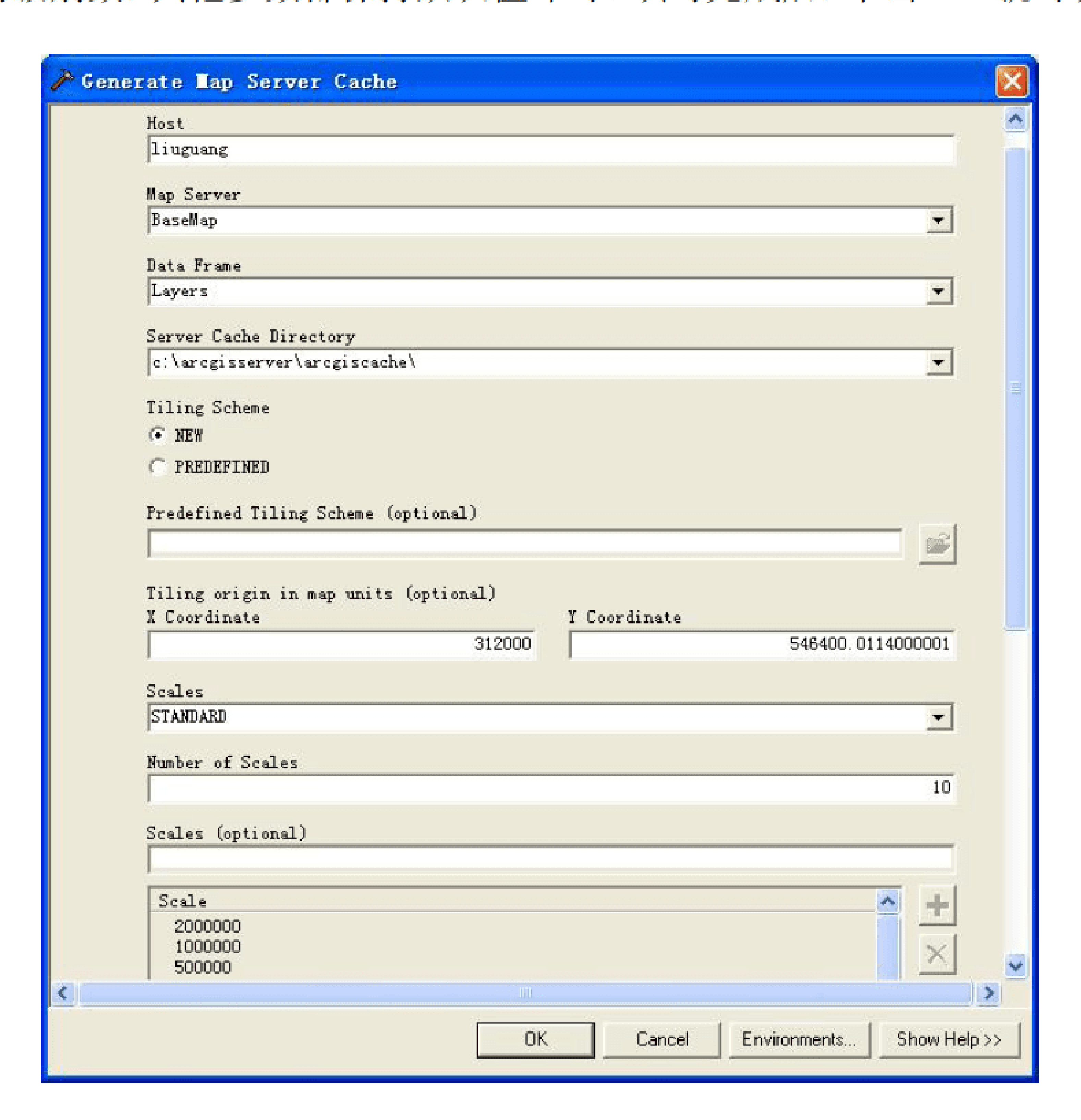


图 3.10　创建地图缓存对话框

3.3 配置文件的使用

服务器依据一组配置文件来维护 GIS 服务器及其服务的配置属性。当服务对象管理器启动时，从这些配置文件中读入配置信息。这些配置文件使用的是 XML 来定义配置属性，包括服务对象容器计算机、输出路径的位置、服务配置属性等等。

GIS 服务器的配置文件为 Server.dat，该文件位于服务对象管理器计算机的 ArcGIS 安装路径的 Server/System 文件夹中。服务配置文件的名称为：

<服务名>.<服务类型>.cfg

它位于 ArcGIS 安装路径的 Server/user/cfg 文件夹中。例如名为 Beijing 的二维地图服务的配置文件名为 Beijing.MapServer.cfg。

当通过 Manager 或 ArcCatalog 等管理界面，或通过调用服务器 API 编写的程序，来设置或修改服务器或其服务的属性时，其实修改的就是相应的配置文件。

由于配置文件都是 XML 格式，因此可以通过文本编辑器手工增加、删除以及修改这些配置文件。增加或删除配置文件，以及修改配置文件后，服务器不能立即反应，只有当服务对象管理器重新启动时才能发现。如果使用 ArcCatalog 来修改，也只有刷新服务器以后才能看到变化。

虽然可以手工来增加、删除与修改配置文件，但是建议还是使用 Manager 或 ArcCatalog 等管理界面来完成。这样如果在服务器配置文件中有错误，那么服务对象管理器会在日志中记录该错误，并使用默认值来替换不正确的属性。如果存在一个错误的服务配置文件，那么服务对象管理器也会记录一个警告，并忽略该配置文件。

3.3.1 服务器配置文件

Server.dat 是服务器配置文件，文件中保存服务器的属性。当服务对象管理器启动时，需要读入该文件的内容。如果成功读入该文件，并完成了文件中指定的每一个具体的初始化，服务器就会报告成功启动。如果文件中有错误，那么服务对象管理器会在日志中记录错误，并使用默认值来代替不正确的属性。

当服务器初次安装服务对象管理器时，Server.dat 文件并不存在。只有服务对象管理器启动以后，或者在 GIS 服务器加入服务对象容器或服务器路径时，Server.dat 文件才会被创建。

下面是笔者 GIS 服务器的 Server.dat 文件内容。该内容表明该 GIS 服务器包含一个服务对象容器（liuguang），一个缓存路径、一个任务路径与一个输出路径，日志级别为 3。

```
<Server>
    <ServerMachines>
        <Machine>
            <Name>liuguang</Name>
            <Description></Description>
            <Capacity>-1</Capacity>
        </Machine>
```

```
    </ServerMachines>
    <ServerDirectories>
        <Directory>
            <Path>c:\arcgisserver\arcgiscache</Path>
            <URL>http://liuguang:8399/arcgis/server/arcgiscache</URL>
            <Description></Description>
            <Type>cache</Type>
        </Directory>
        <Directory>
            <Path>c:\arcgisserver\arcgisjobs</Path>
            <URL>http://liuguang:8399/arcgis/server/arcgisjobs</URL>
            <Description></Description>
            <Type>jobs</Type>
            <Cleaning>sliding</Cleaning>
            <MaxFileAge>21600</MaxFileAge>
        </Directory>
        <Directory>
            <Path>c:\arcgisserver\arcgisoutput</Path>
            <URL>http://liuguang:8399/arcgis/server/arcgisoutput</URL>
            <Description></Description>
            <Type>output</Type>
            <Cleaning>sliding</Cleaning>
            <MaxFileAge>600</MaxFileAge>
        </Directory>
    </ServerDirectories>
    <Properties>
        <LogPath>C:\Program Files\ArcGIS\server\user\log\</LogPath>
        <LogLevel>3</LogLevel>
        <LogSize>10</LogSize>
        <ConfigurationStartTimeout>300</ConfigurationStartTimeout>
        <EngineContextTimeout>600</EngineContextTimeout>
    </Properties>
</Server>
```

3.3.2　服务配置文件

服务配置属性保存在 GIS 服务器的 cfg 文件夹中，每一个配置对有一对应的文件。当在 GIS 服务器中增加一个服务配置时，就自动创建一新的配置文件。当某一配置被删除时，其配置文件从 cfg 文件夹中也被删除。

第 4 章

简单Web GIS应用开发

ArcGIS Server 提供了强大的应用程序开发框架，可满足开发功能强大的企业级地理信息系统。本章将主要介绍如何创建简单的 Web GIS 应用程序以及相关的基础知识。

通过本章你将了解到：

4.1　创建 Web GIS 应用的几种方法
4.2　关于 Web 应用程序框架
4.3　部分页面刷新的实现——Ajax
4.4　自定义工具与命令
4.5　属性查询图形
4.6　右键菜单与地图图片保存

4.1　创建 Web GIS 应用的几种方法

ArcGIS Server 提供了几种可选方法来帮助程序员开发 Web GIS 应用程序。最简单的方法是利用 Manager 工具来创建，其次是使用与 Visual Studio 2005 集成的模板创建，最灵活但是要求最高的是直接使用 Web 控件创建。

4.1.1　使用 Manager 工具创建

创建应用最简单的方法是先使用 Manager 工具中的向导来创建一个 Web GIS 应用，然后在 Visual Studio 2005 中针对具体的需要进行修改。

使用 Manager 工具创建应用的基本步骤如下：

按照本书中 3.1.1 节中介绍的方法，使用 agsadmin 组中的账户登录到 Manager。切换到 Applications 选项卡，然后点击 Create Web Application 链接，进入创建 Web 应用向导的第一个页面。

在第一个页面中，Host 对话框里面已经填写了值，默认为 GIS 服务器的计算机名。需要填写的是 Name，即应用的名称，我们这里输入 NorthAmericaGis。然后选择 Next 按钮进入向导的第二个页面。

在第二个页面中，需要选择地图中要包含的图层。如果是第一次使用该向导，那么首先需要点击 Add GIS Server 链接，加入 GIS 服务器。界面如图 4.1 所示。

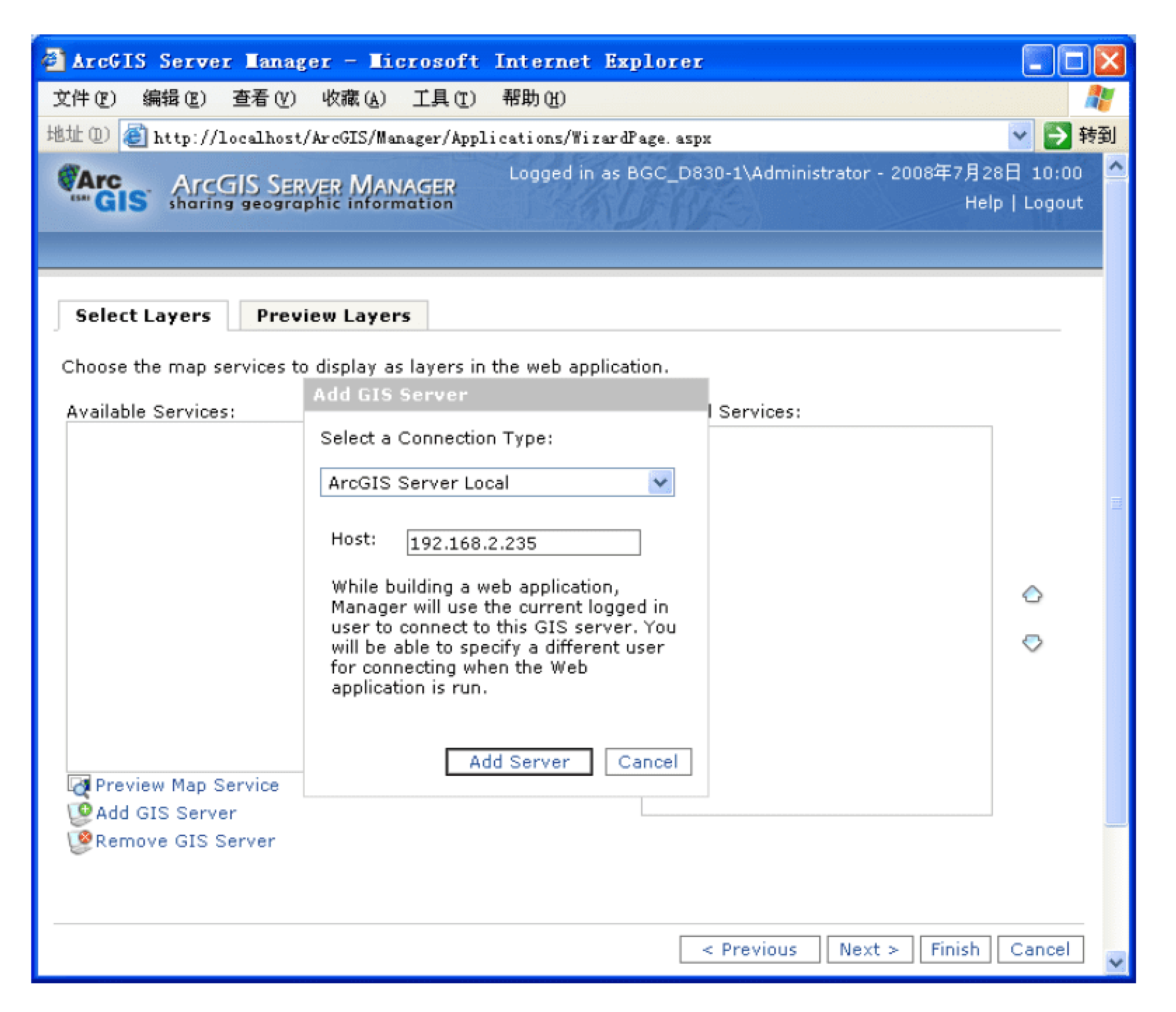


图 4.1　增加 GIS 服务器向导

在该向导中，首先选择连接方式，我们这里选择 ArcGIS Server Local，然后在 Host 中输入要

连接的 GIS 服务器的计算机名或 IP 地址。选择 Add Server 按钮，将该 GIS 服务器添加到可访问的服务列表框中。展开该 GIS 服务器以后，可看到该服务器所发布的服务。如果读者完成了本书中的 3.2.2 节中示例，那么就会看到名为 NorthAmericaMap 的服务。选择该服务，使用 Add 按钮将该服务加入到选择的服务列表框中。在该页面中，还可切换到 Preview Layers 选项卡中，查看地图效果。完成选择服务后，选择 Next 按钮进入向导的第三个页面。

在第三个页面中，需要选择任务。为了简单起见，先不选择任何任务，直接选择 Next 按钮进入向导的第四个页面。

在第四个页面中，需要设置连接服务器的用户。默认已经选择为登录 Manager 工具的用户，一般可满足需要。因此可直接选择 Next 按钮进入向导的第五个页面。

在第五个页面中，需要设置 Web 应用程序的主页面的属性，包括应用程序标题、主色调以及页面上的链接。我们这里将标题设置为“北美洲地图演示系统”，然后选择 Next 按钮进入向导的第六个页面。

在第六个页面中，需要设置应用程序包含的一些地图要素，包括图层控制、鹰眼、工具栏、指北针与比例尺。默认都包含，因此可直接选择 Next 按钮进入向导的最后一个页面。

在向导的最后一个页面中，总结了应用程序的一些基本信息。确保选择了“在新窗口中显示应用程序”。然后选择 Finish 按钮，将在一新的浏览器窗口中显示该 Web 应用程序的运行结果。

由于我们这里使用的是局域网连接，因此应用程序会由于没有身份验证信息，而不能正确显示地图，需要利用 Visual Studio 2005 对源代码进行编辑。

启动 Visual Studio 2005。在开始页面中选择打开 Web 站点，然后在对话框的左边选择 Local IIS，从右边站点列表中选择 NorthAmericaGis（该站点源代码位于 IIS 的 NorthAmericaGis 路径中，默认为 c:\inetpub\wwwroot\NorthAmericaGis）。

打开站点后，在解决方案窗口中右击该站点名称，然后选择 Add ArcGIS Identity 命令，将打开如图 4.2 所示的对话框。在该对话框中输入用户名、口令以及 GIS 服务器的计算机名。

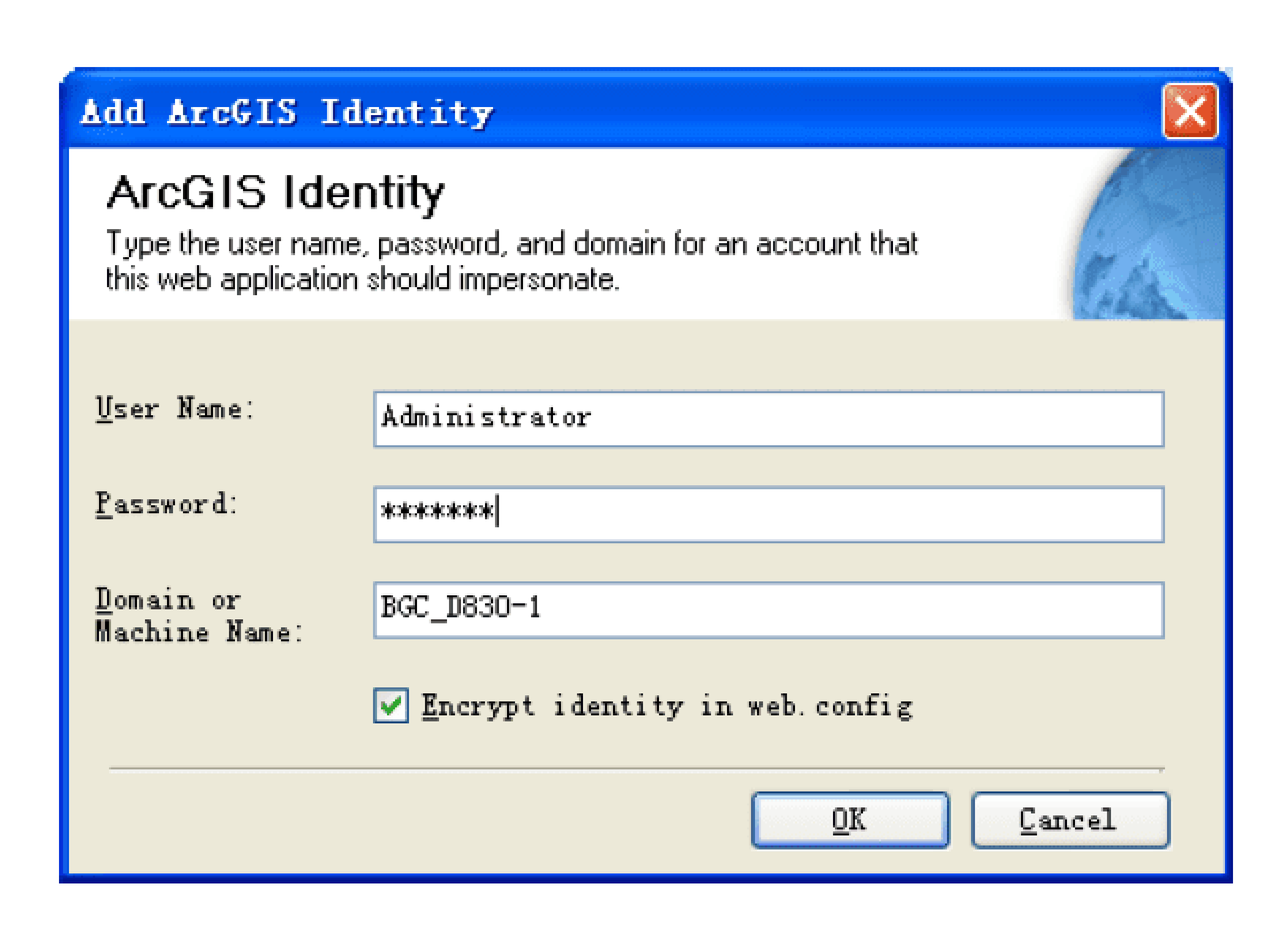


图 4.2 “添加 ArcGIS 身份”对话框

添加完成以后，运行程序，便可得到正确的结果，如图 4.3 所示。

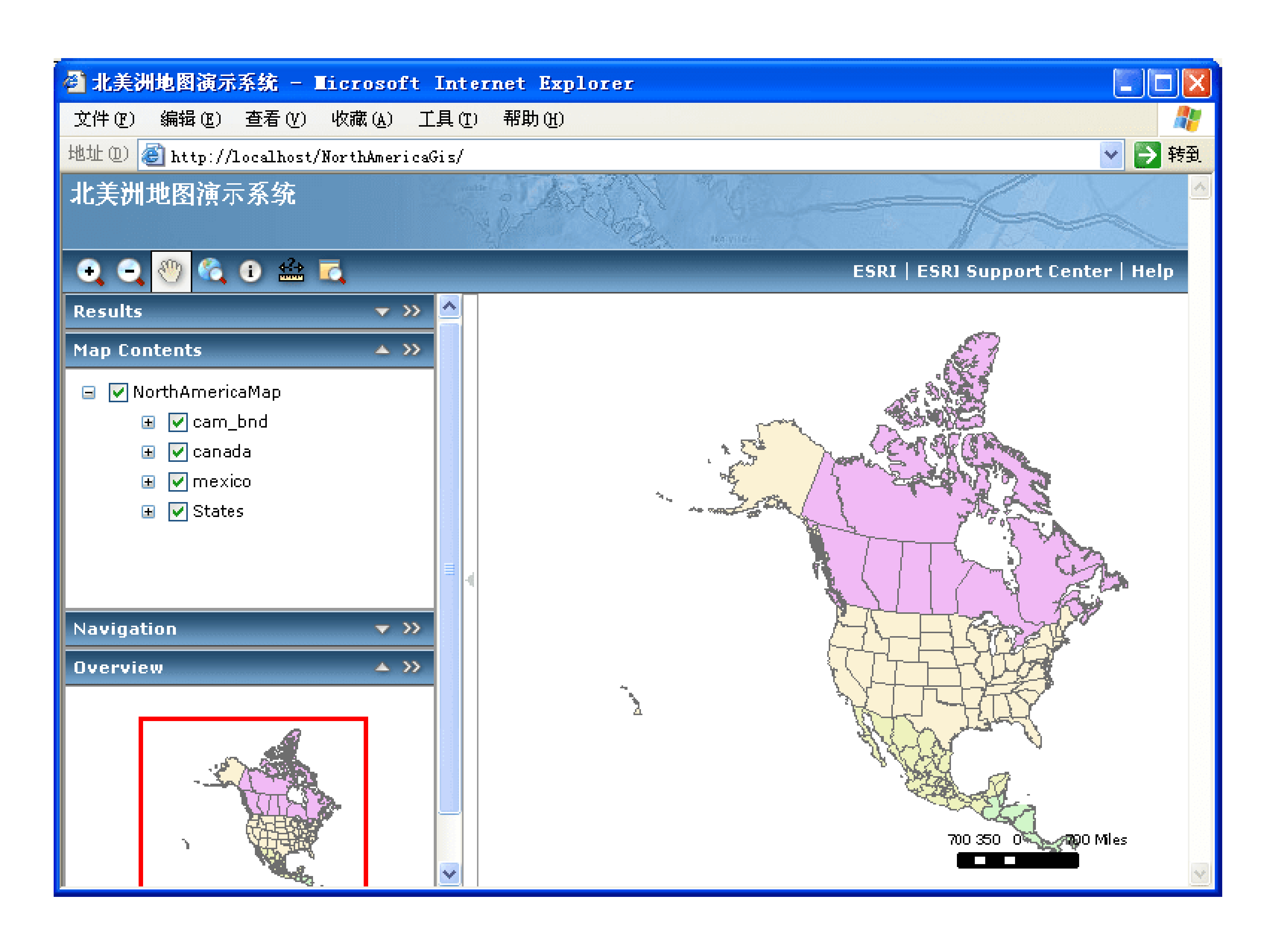


图 4.3　通过 Manager 工具创建 Web 应用的结果

4.1.2　使用 Visual Studio 模板创建

ArcGIS Server 安装时，会在 Visual Studio 2005 中安装一 Web 应用模板。该模板就是 Manager 工具创建 Web 应用的模板。

启动 Visual Studio 2005 后，在开始页面中选择创建 Web 站点，打开 New Web Site 对话框（图 4.4），在该对话框的模板中选择“Web Mapping Application”。站点位置选择为 HTTP，站点名称为 NorthAmericaSample，语言为 Visual C#。

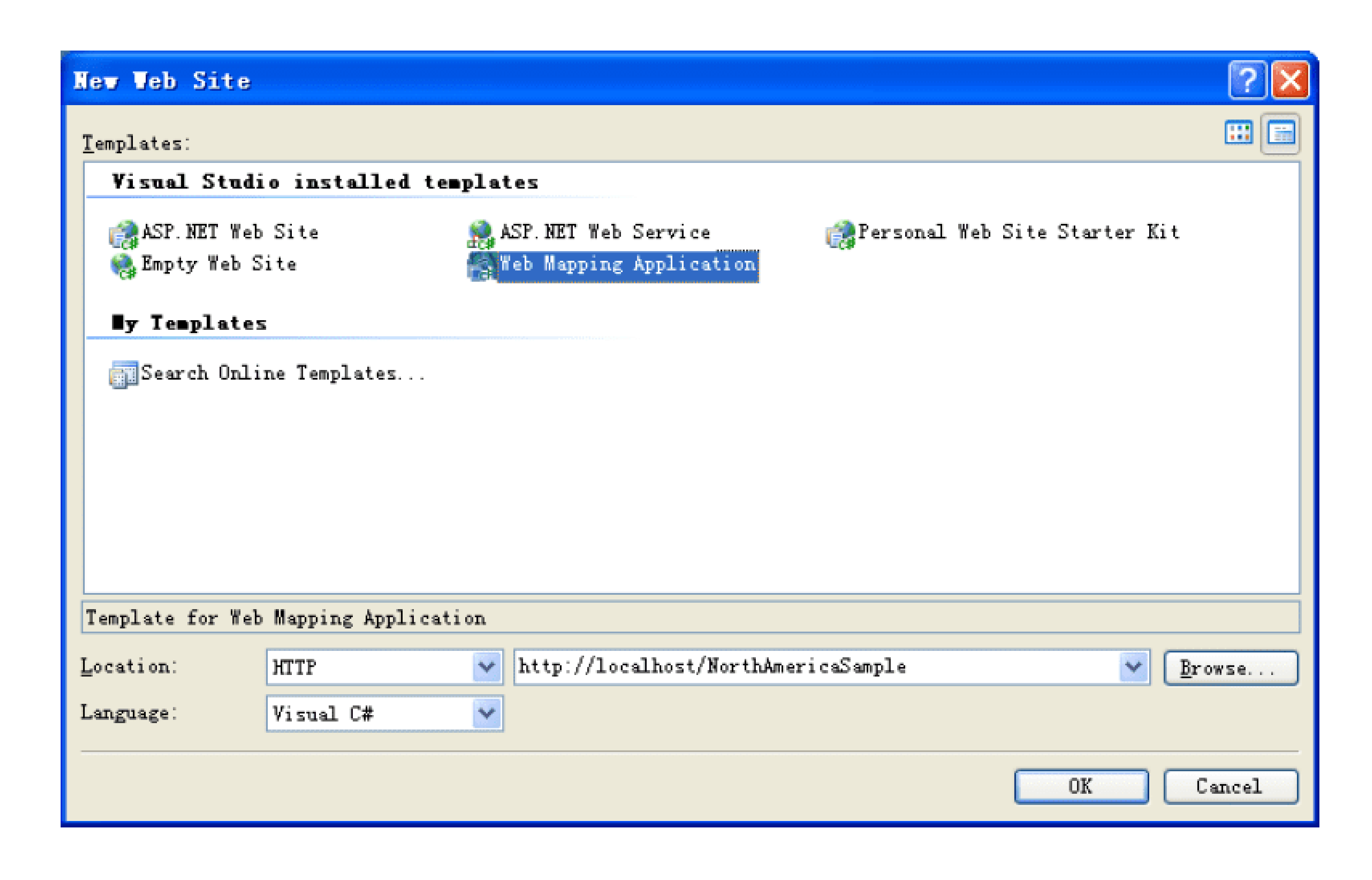


图 4.4　使用模板创建 Web 应用

选择 OK 后，将创建包含图 4.5 所示内容的站点，与利用 Manager 工具创建的 Web 应用程序

所包含的内容是一致的。

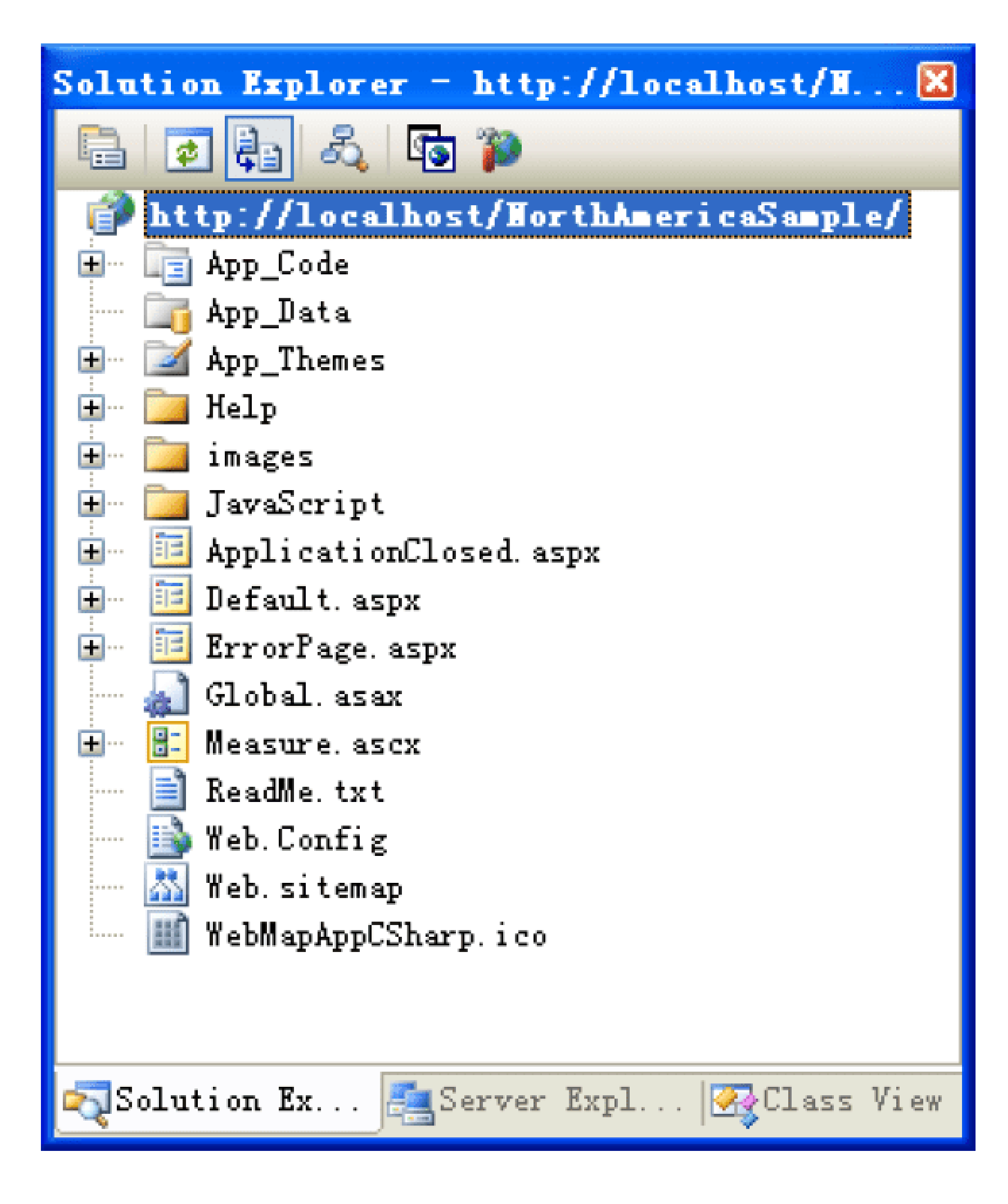


图 4.5　利用模板创建的 Web 应用程序所包含的内容

- Default.aspx 页面是主页面，包含了地图及其相关联的控件与内容；
- ErrorPage.aspx 是错误页面，当应用程序遇到未处理的错误时显示该页面；
- ApplicationClosed.aspx 页面是当用户单击关闭链接（只有使用非池化 ArcGIS 服务器数据源时才会显示）时显示的页面。这样可允许程序释放在 GIS 服务器上使用的资源；
- Measure.ascx 是一自定义用户控件，该控件在 Default.aspx 中使用，用于在地图上测量距离与面积；
- Web.config 是一标准的 ASP.NET 配置文件，里面存储了.NET 配置信息，以及当使用 ArcGIS Server Local 数据源时的身份信息；
- Web.sitemap 是 ASP.NET 2.0 站点地图配置文件。Default.aspx 中的 SiteMapDataSource 控件使用该文件在 Menu 控件中显示其中的链接；
- ReadMe.txt 中包含了在 Visual Studio 中配置 web 应用程序的简单描述；
- App_Code 是一标准的 ASP.NET 文件夹，用于存放代码，其中包含了 MapIdentify.cs 文件，该文件用于点查询工具。App_Data 和 App-Themes 也是标准的 ASP.NET 文件夹。App-Data 用于存放应用程序的数据，通常是 SQL Server Express 数据库。App_Themes 用于存放不同主题的文件，包括样式表单、皮肤文件以及图片。Help 文件夹中包含了帮助文件。images 文件夹包含了程序使用的图片。JavaScript 文件夹包含了程序使用的 JavaScript 库文件。

该 Web 应用还需要使用存放在 IIS 根路径中 aspnet_client 文件夹中的 ASP.NET 公用文件，这些文件通常在 C:\Inetpub\wwwroot\aspnet_client\ESRI\WebADF 中。该文件夹中包含了 ESRI 的 Web 控件使用的图片、JavaScript 库以及样式表单。Web 应用程序必须要能访问这些文件才能正常工作。

Default.aspx 是 Web 应用中主要显示地图的页面。该页面包含了显示地图的 Web 控件以及相关的辅助控件，例如图层控制、鹰眼以及与地图交互的工具。默认包含的控件及其设置如图 4.6 所示。

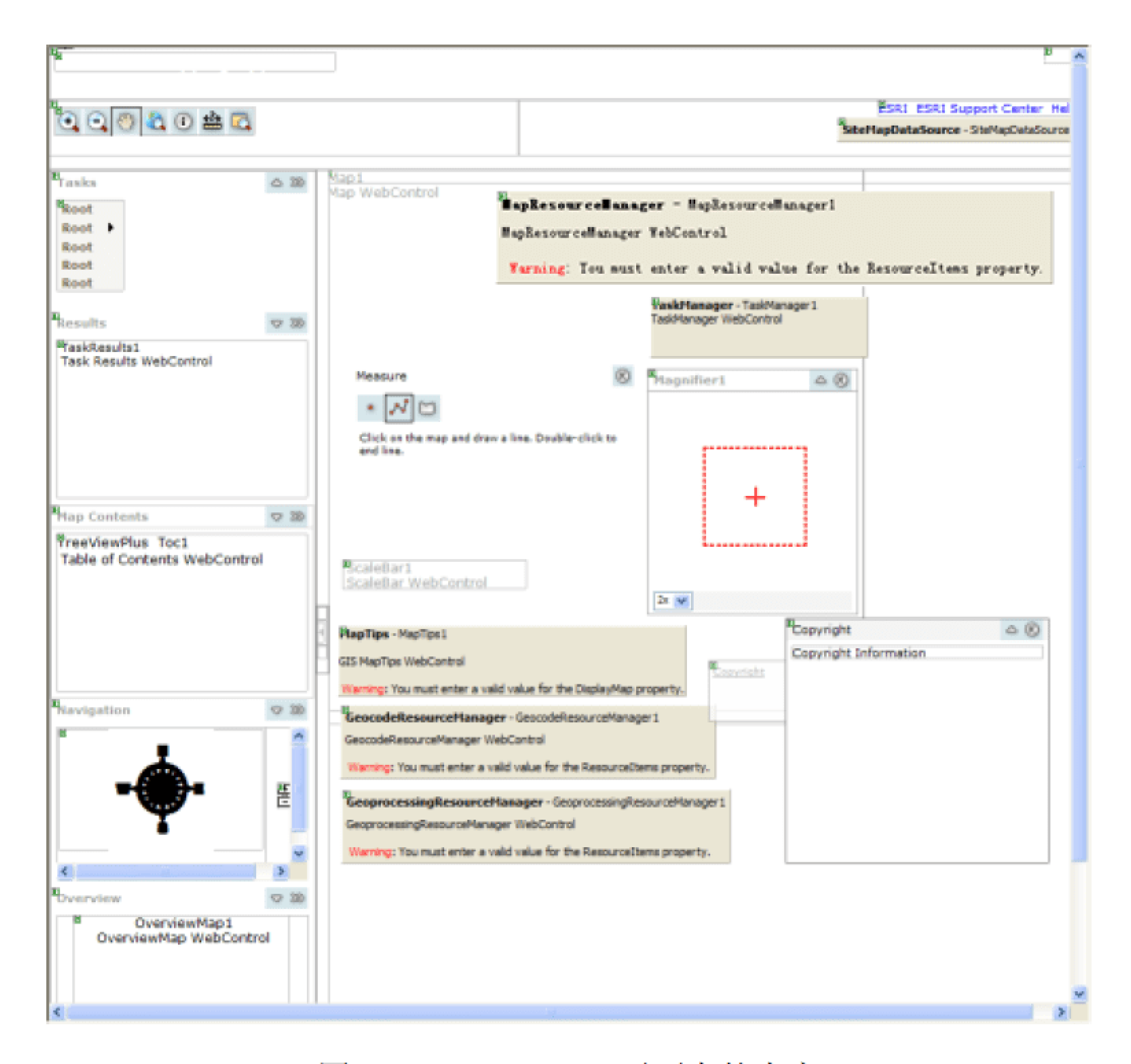


图 4.6　Default.aspx 页面中的内容

Default.aspx 页面中也包含有与服务器通讯的代码，例如获取地图图像切片。通过回调（Callback）技术，该页面通常不用刷新整个页面。页面将消息发送给服务器，然后只刷新相关部分来显示返回结果。例如，当用户使用点查询工具在地图上单击后，页面使用回调，从服务器得到返回结果，并将结果显示在任务结果面板中，而不是使用回发（Postback）导致整个页面刷新。

当使用模板创建 Web 应用后，第一个要设置的就是 MapResourceManager 控件的 ResourceItems 属性。从 MapResourceManager 控件的名称就可以看出该控件是用于管理地图资源的，页面中地图控件的显示内容由该控件管理。

首先切换到 Default.aspx 页面的设计视图，选择 MapResourceManager 控件，然后在属性页面中单击 ResourceItems 右侧的省略号，打开地图资源项集合编辑器，如图 4.7 所示。

在地图资源项集合编辑器中，默认时其内容为空。选择 Add 按钮增加一 MapResourceItem 对象。该对象中包含了几个属性，这些属性用于控制该地图资源的数据源如何被应用程序中的地图、图层控制等其他控件应用。

其中 Name 属性用于在应用程序中唯一标识该资源。该名称作为地图服务显示在图层控制控件中。默认名称一般是 MapResourceItem 加序号，一般建议修改为地图服务的名称。我们这里设置为 NorthAmericaMap。

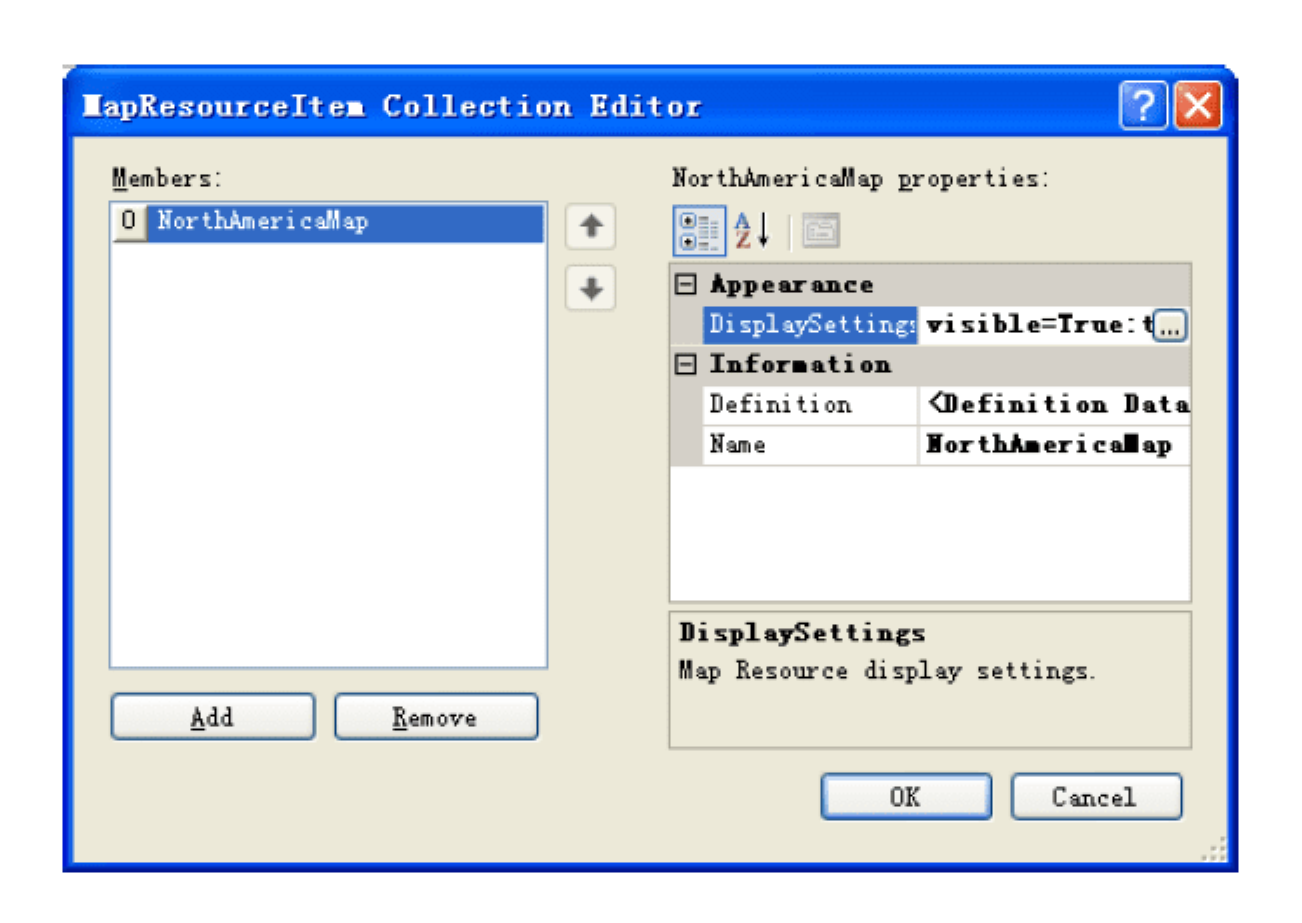


图 4.7 地图资源项集合编辑器

单击 Appearance 属性组下 DisplaySettings 右侧的省略号，可打开“地图资源显示设置编辑器”（图 4.8）。该编辑器用于定义地图资源生成图片的内容。

透明颜色与背景共同决定了地图图片的透明绘制。融合属性定义该地图图片相对于其他资源的地图图片的可见性。当地图中同时显示几个地图资源中的地图时，每个地图资源所生成的图片需要融合为一张图片。在资源列表中最下面的地图资源最先绘制，然后是上面的地图资源绘制。透明值（Transparecy）用于定义该地图资源所生成的图片的透明程度。0%表示图片不透明，100%表示完全透明，也就是不可见；界于这两者之间的值表示可显示位于该地图下面的其他地图资源的内容。当图片部分透明时，融合所有地图资源的图片就会花费更长的时间。需要 MIME 数据（Recuest MIME Data）选项确定如何从数据源请求地图图片。如果数据源支持以 MIME 格式发送地图图片，那么该图片就会被应用程序保存在内存中。如果不支持或没有选择 MIME 数据，那么数据源必须先在服务器的输出路径中生成一图片文件，然后与应用程序共享该文件所在文件夹。图片格式（Image Format）确定了数据源生成的图片的格式。可见选项（Visible）确定是否生成地图图片。在图层控制中显示选项（Display）允许隐藏资源。该资源在地图或鹰眼控件中仍可显示，也还可被 SearchAttributesTask 等其他控件利用。

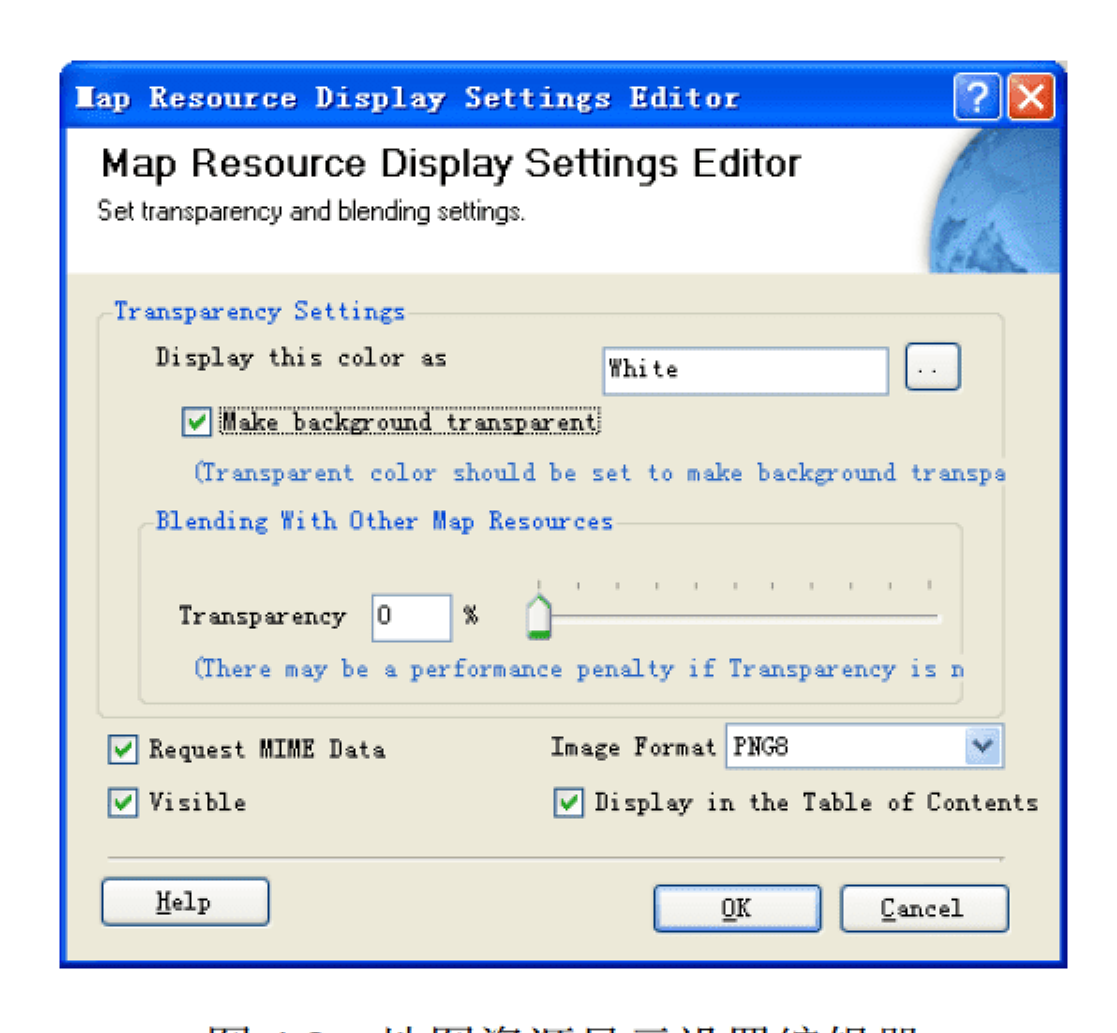


图 4.8 地图资源显示设置编辑器

在地图资源项集合编辑器中，Definition 属性提供了一系列对话框来连接数据源提供者（例如 GIS 服务器）与创建地图资源。单击 Definition 右旁的省略号，打开资源定义编辑器。在该编辑器中，首先要确定数据源的类型，不同类型需要不同的设置。

最常用的是 ArcGIS Server Local 类型，这也是我们这里要详细说明与使用的类型。选择好类型后，需要设置数据源。如果程序员开发使用的计算机就是 GIS 服务器，那么可设置为 localhost；如果要设置为 GIS 服务器的是局域网内其他计算机，设置为计算机名或 IP 地址均可。一旦设置好数据源以后，Resource 属性打开 ArcGIS 资源定义编辑器，通过该编辑器可选择服务器中的某个地图服务。这里我们选择第 3 章创建的 NorthAmericaMap 地图服务。设置过程如图 4.9 所示。

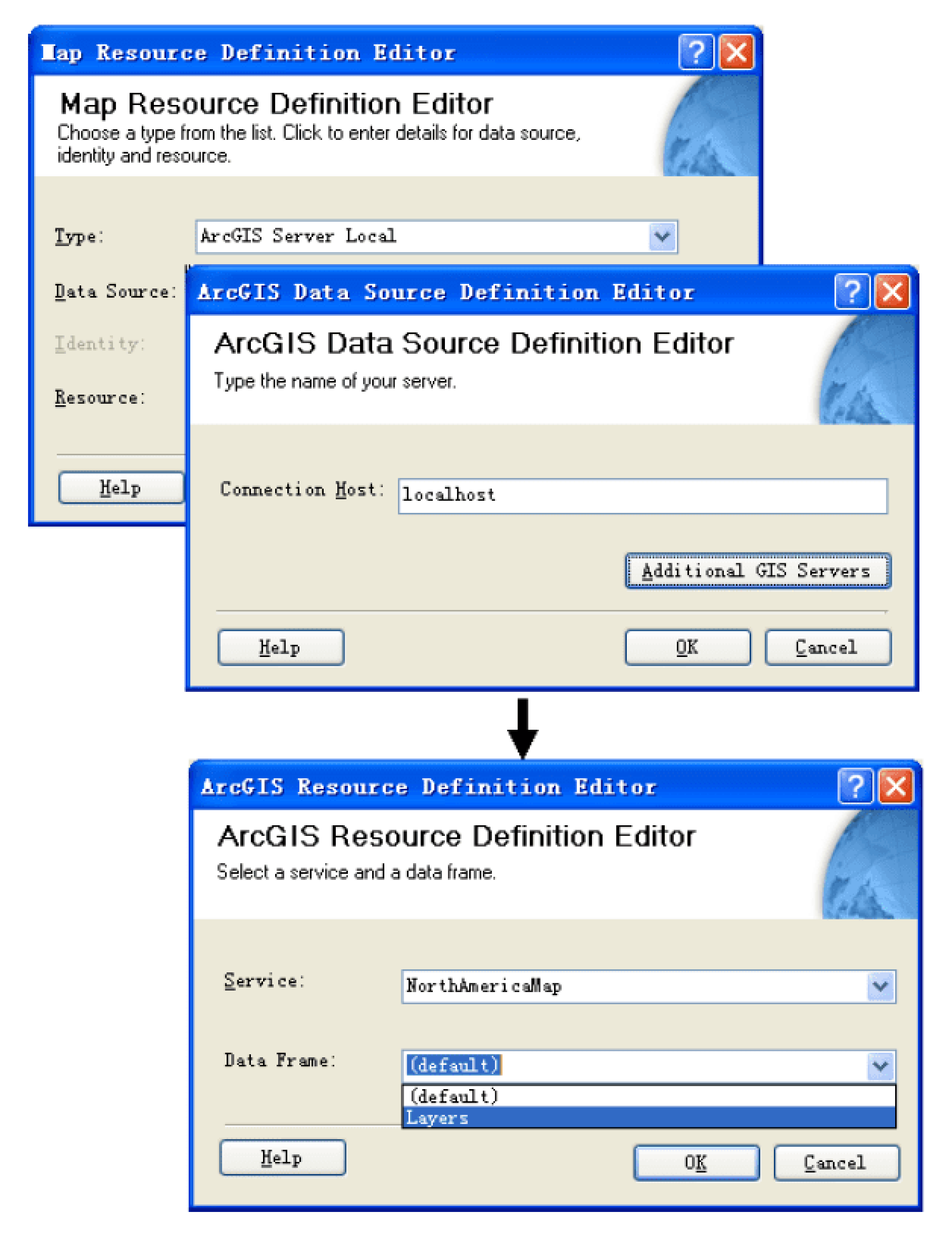


图 4.9 设置 ArcGIS 数据源作为地图资源

这时便可编译并运行程序，运行结果与图 4.3 一致。

在地图资源定义编辑器中 Identity（身份）属性为灰色，表示不可设置。在设计阶段使用运行 Visual Studio 用户的身份去连接 ArcGIS 服务器的本地数据源，在运行期间，Web 应用程序会自动建立该身份。在一个 Web 应用程序中只能使用一个身份来访问 ArcGIS 服务器本地数据源。添加 ArcGIS 身份的方法可按照 4.1.1 节中的介绍。

4.1.3 使用 Web 控件创建

有时如果系统比较简单（为了便于读者理解，本书的大部分实例使用该方式，因此读者需要很

好的理解该节介绍的步骤，后面将不再赘述，而只直接引用），我们可以直接使用 ArcGIS Server 的 Web ADF 所提供的控件，而不是使用模板，来创建 Web 应用程序。这是因为模板包含了许多系统不需要代码与资源。

Web ADF 控件集成在 Visual Studio 2005 的可视化设计环境中，可以像使用 Visual Studio 2005 提供的控件（例如文本框、按钮）一样来使用这些控件。

下面的例子演示了如何使用 Web ADF 控件从零开始创建一个 Web 应用程序。步骤如下：

（1）在 Visual Studio 2005 中，选择 File 菜单中的 New Web Site 命令，将弹出 New Web Site 对话框。

（2）在 New Web Site 对话框中，将 Location 的值设置为 HTTP，将 Language 设置为 Visual C#。

（3）在 Visual Studio installed templates 中，选择 ASP.NET Web Site。

（4）输入该 Web 应用的名称与位置，这里设置为 http://localhost/FromControl。然后选择 OK。这时 Visual Studio 2005 将在设计视图中显示一个空白的 Default.aspx 页面。

（5）打开 Visual Studio 的工具箱，并展开 ArcGIS Web Controls 选项卡。分别拖动一个地图资源管理控件（MapResourceManager）与一个地图控件（Map）到 Default.aspx 页面中，保留它们默认的 ID 设置，即分别为 MapResourceManager1 与 Map1。

（6）下面需要设置的是 MapResourceManager 控件的 ResourceItems 属性。过程及设置同 4.1.2 节中有关地图资源管理控件设置内容的介绍。

（7）将地图控件 Map1 的 MapResourceManager 属性值设置为 MapResourceManager1。

（8）调整地图控件 Map1 的大小，例如 300×300、500×500 等像素。

（9）在 Default.aspx 页面中加入一个内容目录控件（Toc，Table of Contents 的缩写），保留其 ID 的默认设置，即为 Toc1。

（10）将控件的定位改变为绝对值定位，这样才能使控件可放置到页面中的任何位置。右击 Toc1 控件，选择右键菜单中的 Style 命令，打开 Style Builder 对话框。在该对话框中切换到 Position 选项卡，然后将 Position mode 设置为 Absolutely position，如图 4.10 所示。选择 OK 按钮退出对话框。现在可在设计视图中将 Toc1 拖动到地图控件的右边。

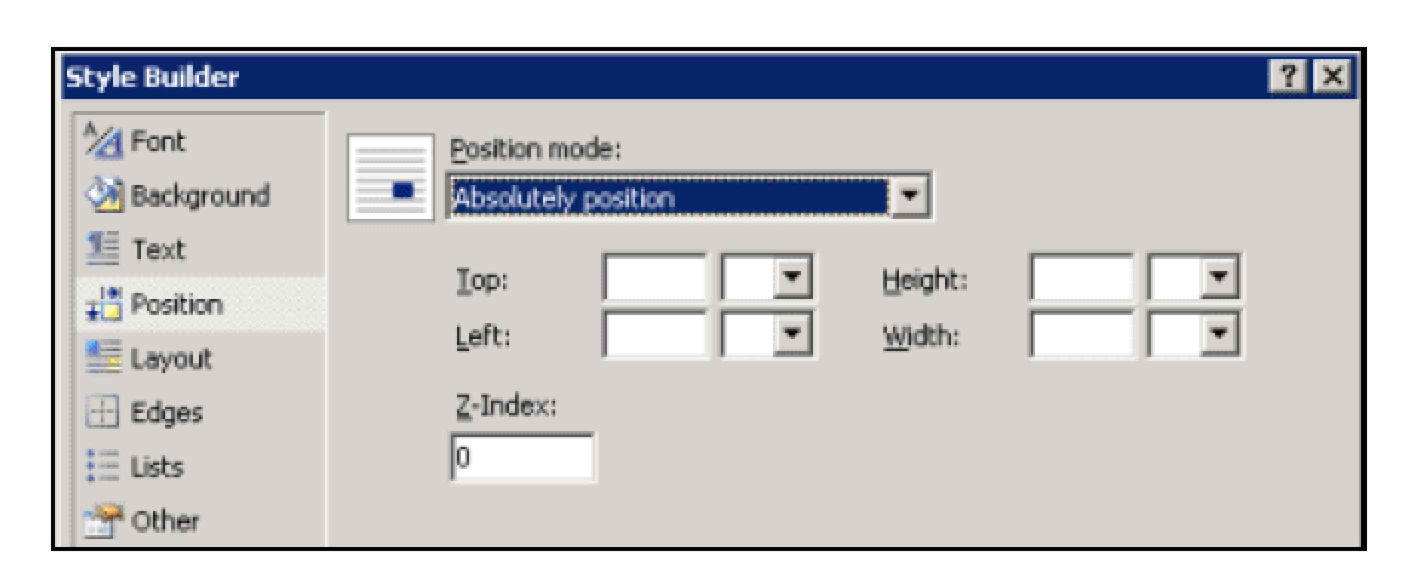


图 4.10 设置定位方式

（11）将 Toc1 控件的 BuddyControl 属性设置为 Map1，这样就可以用 Toc1 控件来控制 Map1 中显示的图层。

（12）在 Default.aspx 页面中加入一个工具条控件（Toolbar），保留其默认的 ID 设置，即为 Toolbar1。

（13）设置 Toolbar1 控件的 BuddyControls 属性。该属性是一连接控件集合，在集合中加入 Map1。

（14）设置 Toolbar1 控件的 ToolbarItems 属性。通过属性视图打开如图 4.11 所示的 ToolbarCollectionEditorForm 对话框。在该对话框，添加 Web ADF 已经封装好的地图操作按钮，包括放大（MapZoomIn）、缩小（MapZoomOut）、漫游（MapPan）以及 MapFullExtent（全图）。加入工具按钮后，选择 OK 按钮退出对话框。

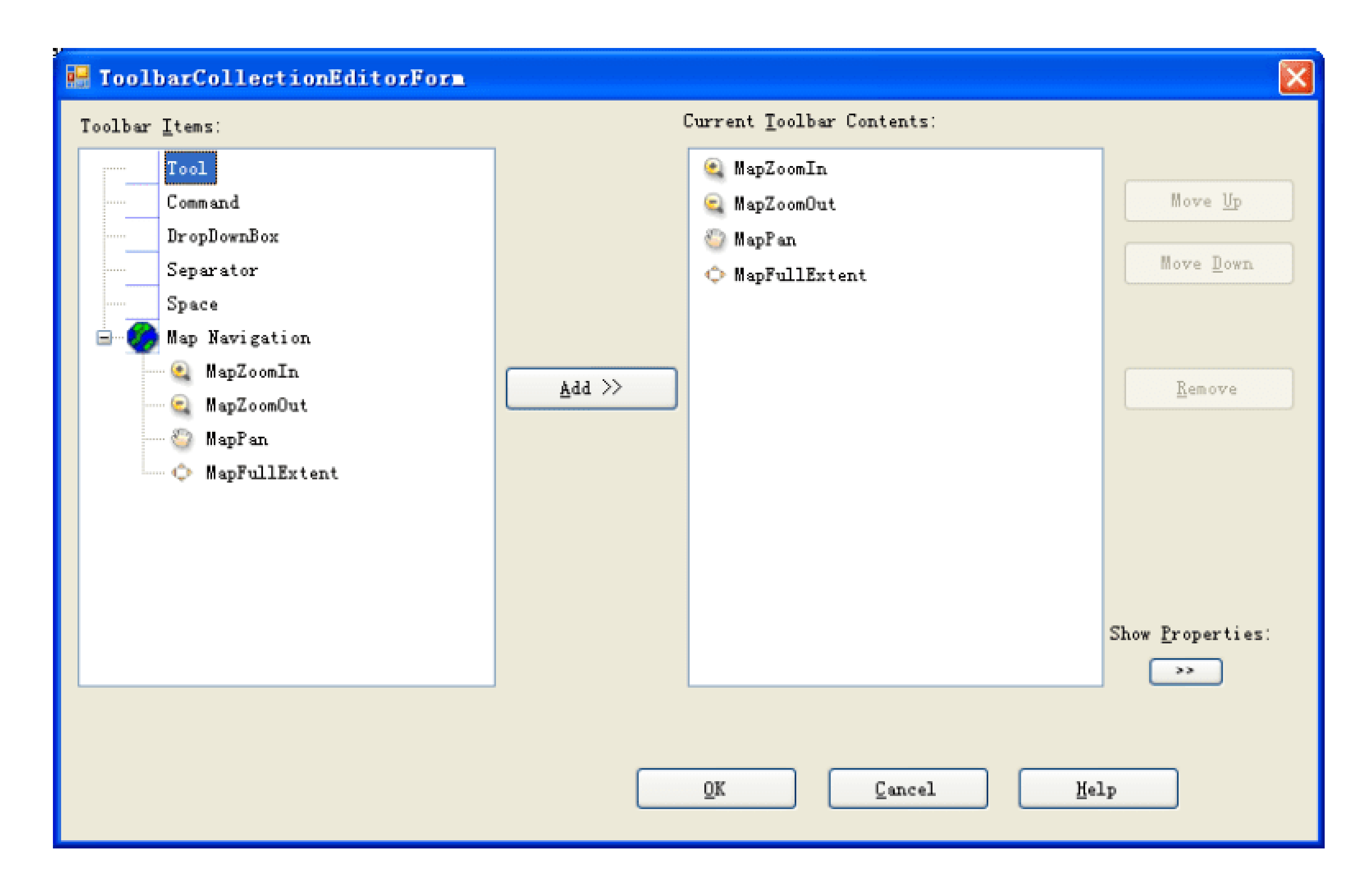


图 4.11　通过 ToolbarCollectionEditorForm 对话框加入工具按钮

（15）按照 4.1.1 节中介绍的方法增加 ArcGIS 身份。

至此我们就通过直接使用 Web ADF 提供的控件，创建了一个简单的 Web GIS 应用程序。运行应用程序，可得到如图 4.12 所示的界面。

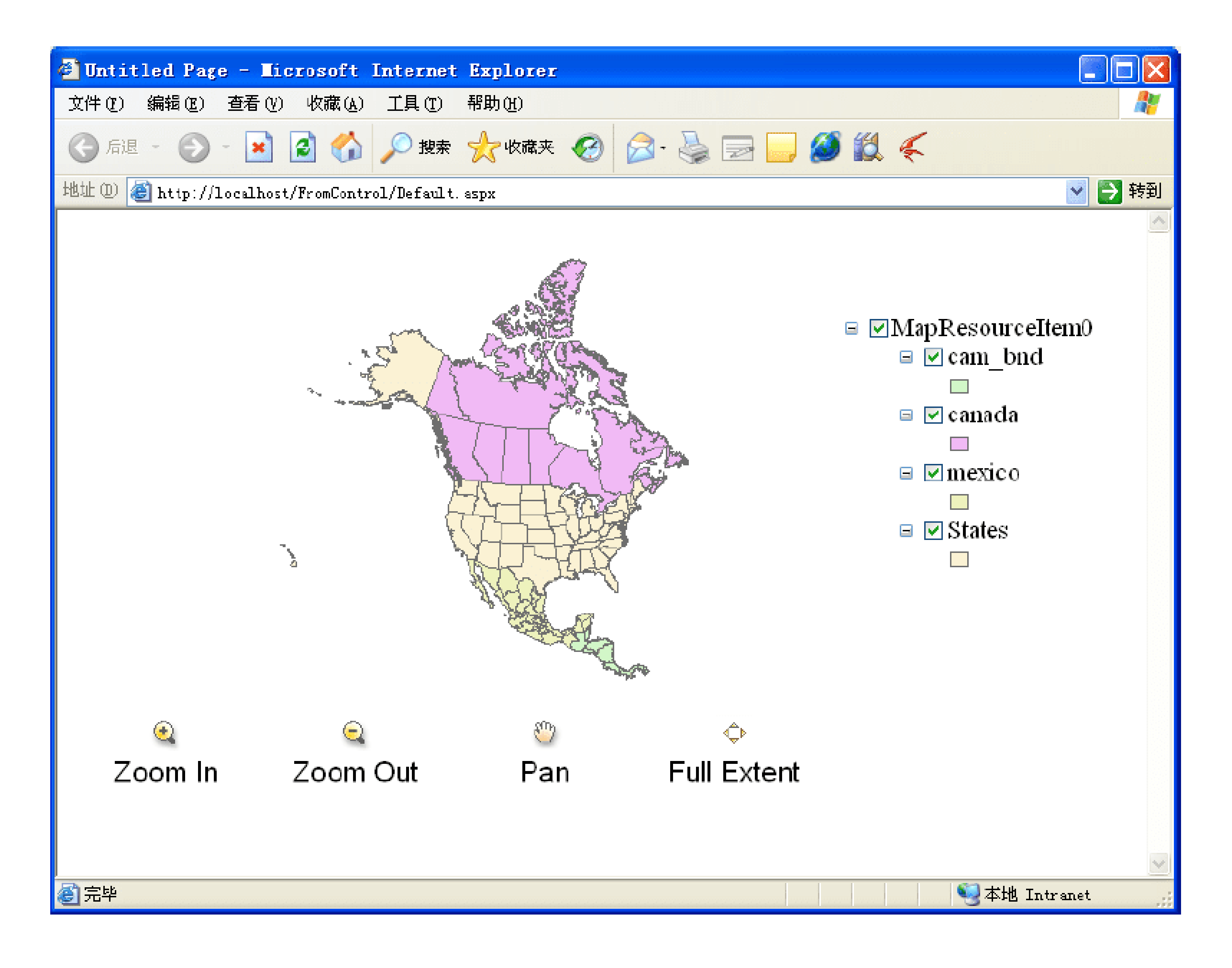


图 4.12　系统运行界面

4.2 关于 Web GIS 应用程序框架

Web 应用程序框架（Web ADF）是 ESRI 为了简化在 Web 上提供如地图浏览这样的 GIS 服务而实现的一个开发框架。

4.2.1 Web 应用程序框架体系结构

Web ADF 与其他的相关组件的关系如图 4.13 所示。

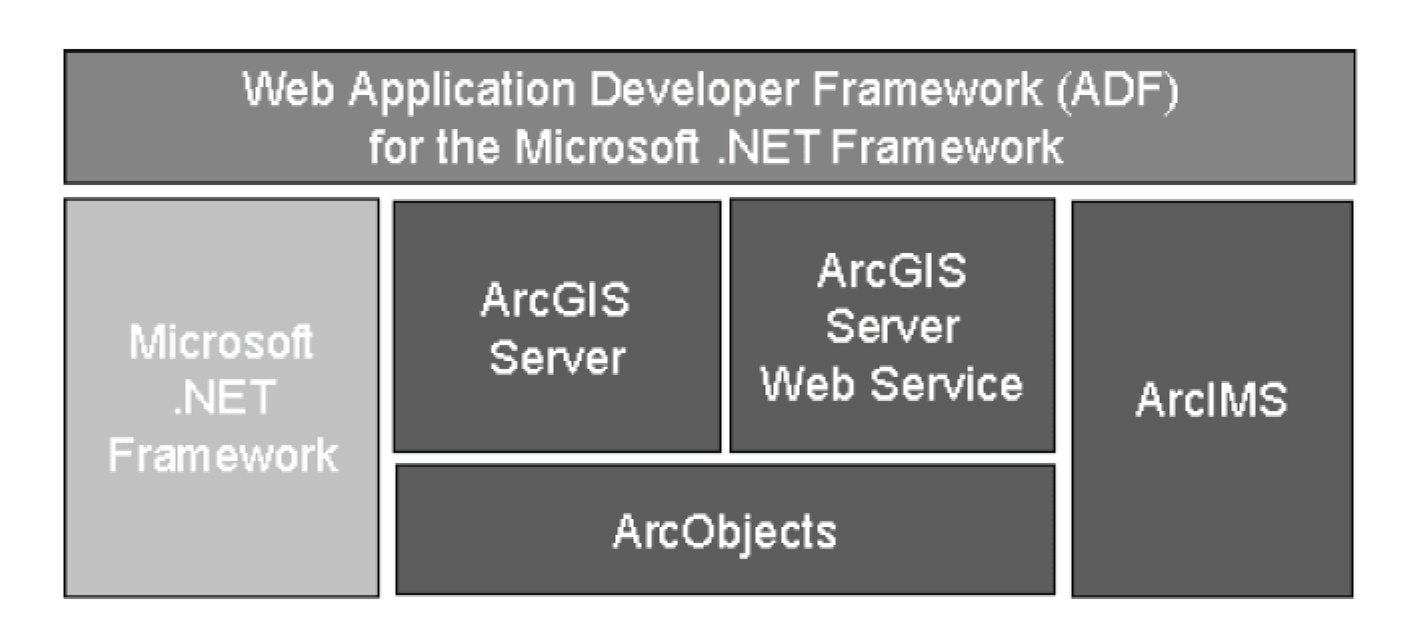


图 4.13 Web ADF 与其他相关组件的关系

从图中可看到，Web ADF 是建立在 Microsoft .NET 框架之上的一些新的类，这些新的类扩展了.NET 框架类库，提供了一系列自定义 Web 控件以及支持本地与远程访问的数据源。图的右部也清楚地表明了 Web ADF 在整个 ESRI 类库中的位置。

以前版本的 ADF 称为 ArcGIS Server ADF，那是因为它只支持单一的数据源，即基于 ArcGISObjects 的 ArcGIS Server。而现在的 Web ADF 在两个方面进行了扩展。一个是当前的 ADF 支持多个数据源，包括 ArcGIS Server 与 ArcIMS 等。另一个是该多数据源架构允许在同一应用程序中同时集成与访问来自不同来源的数据。

一个单独的数据源可以提供几个功能。例如，一个 ArcIMS 地图服务可用于输出地图图片，还可以用于查询要素图层与地理编码。Web ADF 中的 Web 控件是一些显示与访问地理数据的可视化控件。这些控件如何使用数据源依赖于数据源能做什么。

1. Web 控件、资源管理器、资源与功能与之间的关系

那么 Web 控件是如何与数据源连接的呢？答案如图 4.14 所示。控件和数据源之间的关系是通过一系列的资源管理器（Resouce Manager）控件来维护的。资源管理器决定哪些数据源是可以使用的资源（Resouces），以及这些资源怎么被控件所使用。一旦一个数据源被资源管理器管理之后，就被展现为资源。控件通过资源到达数据源。资源可以把数据源以多种形式展现出来，比如可以提供一幅地图展现在地图控件中，它也可以把数据源以图层列表的方式展现在内容目录控件中，也就是说资源拥有不同的能力，这些不同的能力我们称为不同的“功能”（Functionality）。功能定义了资源怎么被使用。

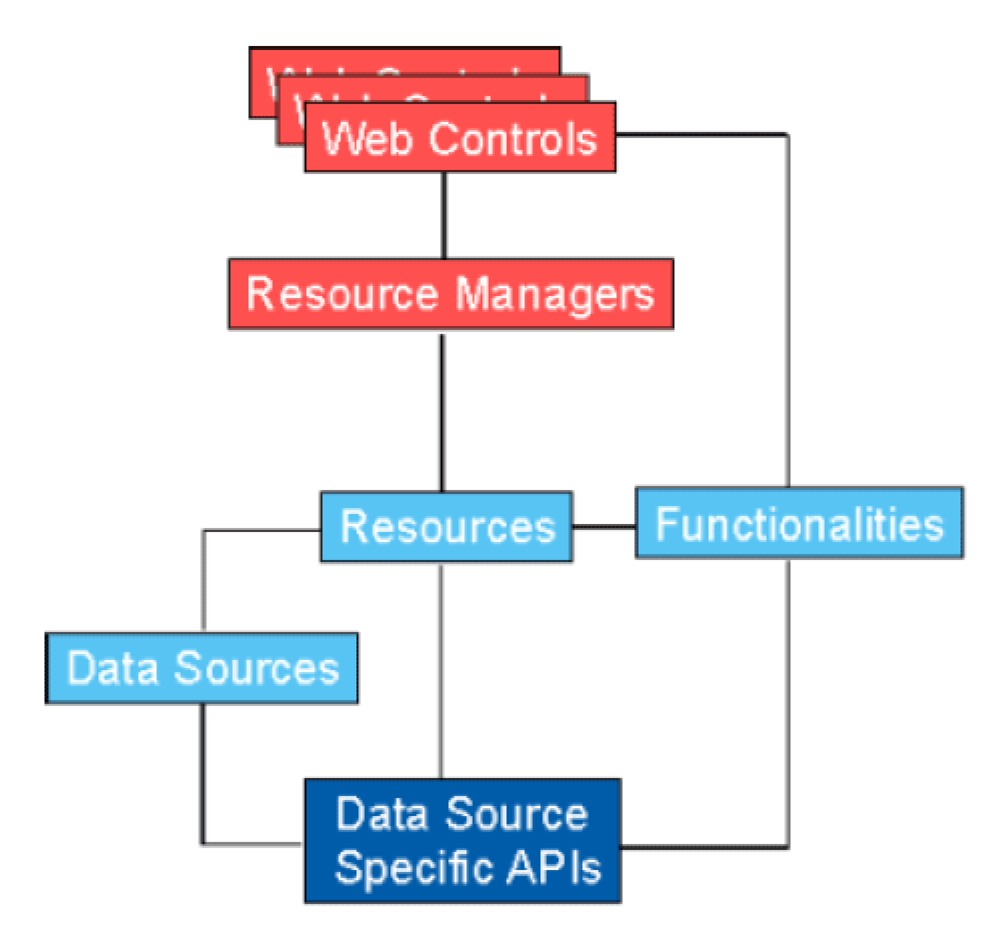


图 4.14　Web 控件与资源管理器、资源、功能、数据源之间的关系

- 从控件角度来讲，不同的控件可以通过不同的方式来使用相同的数据源，比如一个资源可以为地图控件提供一幅地图，也可以为内容目录控件提供一套图层的列表，这就是资源的不同的功能；
- 从数据源角度来讲，不同的资源会通用的展现一些能力，也就是提供不同的功能，比如展现地图，查询地图等。

2. 公有 API 与特有 API 的关系

资源可以展现为不同的能力，但是具体能够展现为哪些能力还是要看数据源本身能够提供什么样的功能。有一些功能是所有数据源都能办到的，也就是说资源可以展现出一些所有的数据源都可以提供的能力，比如提供地图，比如查询地图，无论是 ArcGIS Server 作为数据源，还是 ArcIMS 作为数据源，这些都是基本的能力。因此 Web ADF 就把实现这些基本的普通的功能所需要的类归为公有 API（Common API）。而有些功能是有些数据源特有的，比如提供编辑功能，那是 ArcGIS Server 特有的，ArcIMS 无法提供。这些就被称为特有 API（Specific API）。特有 API 包括 ArcIMS API、ArcWeb API、OGC\WMS API、ArcGIS Server SOAP API 与 ArcGIS Server ArcObject API（图 4.15）。注意其中的 ArcObject API 也被列为了特有 API，它是 ArcGIS Server 数据源的特有 API。

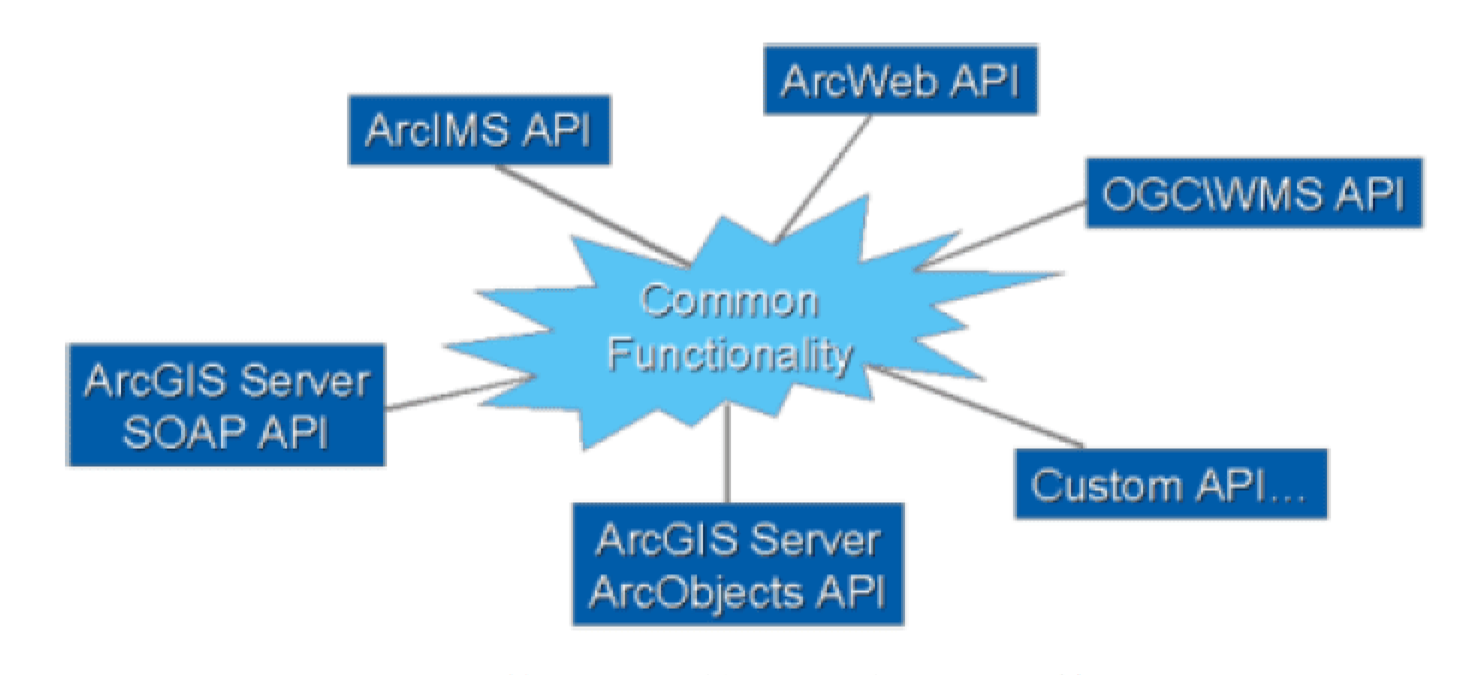


图 4.15　使用不同的 API 访问不同数据源

3. 公有 API 的基本结构

Web ADF 的多源架构的基础是公有 API，它是一包含类与接口的抽象框架。不同的数据源可

通过实现公有 API 来作为插入到 Web ADF 中。

我们在开发过程中首先接触到的也就是这些公有 APIs，主要包含三个接口：

- ❑ IGISDataSouce，定义了数据源的连接。
- ❑ IGISResouce，定义了数据源提供的信息类型等。
- ❑ IGISFunctionality，定义了资源怎么被使用。

这三个接口是不同的数据源可以展现一些基本功能的基本接口，也就是说不同的数据源要实现基本的功能必须实现这三个基本的类，才能在控件上展现出那些基本的能力。各种数据源都用相应的类实现了这三个接口。这三个接口之间的关系如图 4.16 所示。

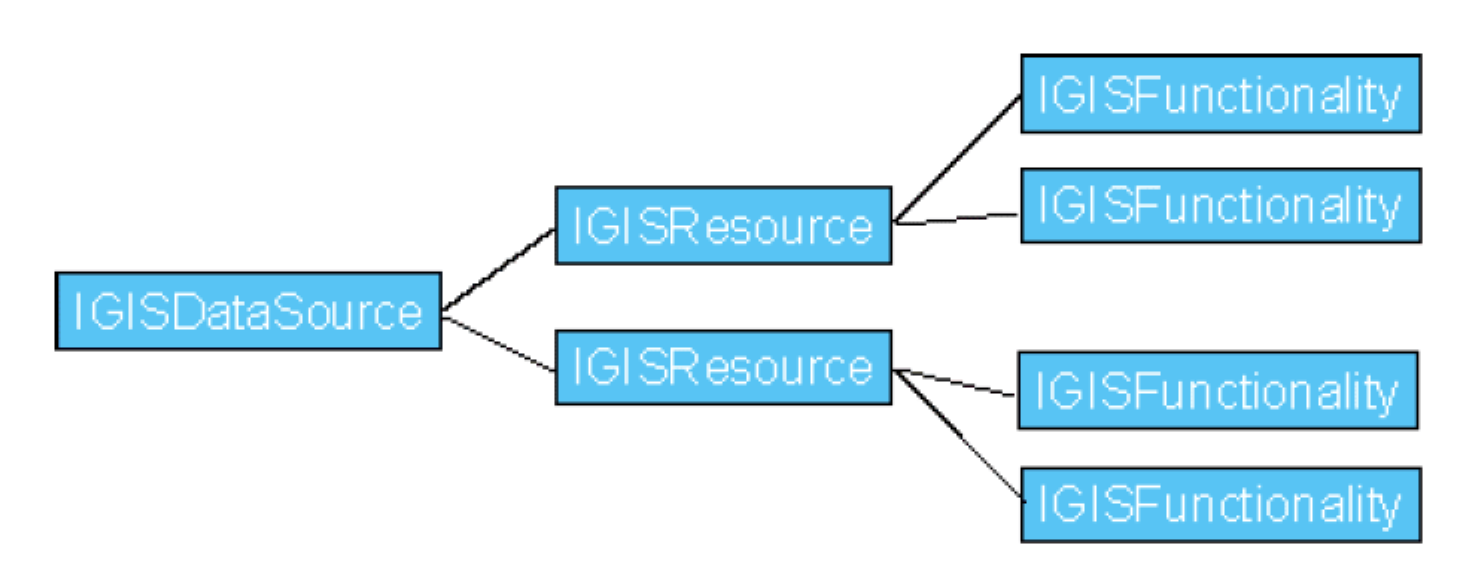


图 4.16 公有 API 中主要三个接口的关系

一个数据源（GISData Source）会包含一系列的资源（GISResource）。以 ArcGIS Server Local 而言，它包含了 MapResouce、GeocodeResouce、GeoprocessingResouce 等几种资源。一个资源又会包含一系列的功能（GISFunctionality）。功能主要有两类：MapFunctionality 与 QueryFunctionality。MapFunctionality 主要展现资源的地图能力，比如输出地图图片，改变地图范围，设置地图中图层的可见性等。而 QueryFunctionality 主要展现资源的数据的空间和属性查询能力。

4. Web ADF 中如何使用公有 API？

如上所述，不同的数据源都有相应的类来实现上面的基本接口，那我们来看看 ArcGIS Server Local 数据源的相应的实现类：

- ❑ IGISDataSouce: GISDataSouceLocal
- ❑ IGISResouce: MapResouceLocal、GeocodeResouceLocal
- ❑ IGISFunctionality: MapFunctionality、QueryFunctionality

这就是 Web ADF 的优势所在，它可以使得各种不同的数据源都展现为资源，使得它们可以以相同的方式得到使用。对于控件而言，每个资源就像一个图层，而不管数据源是什么。

就举一个地图放大的功能，一个地图控件中有两个数据源，一个是 ArcGIS Server Local，一个是 ArcIMS，地图的范围重新设定了之后，控件都通过每个资源提供的 MapFunctionality 给每个资源重新设定范围，每个资源输出这个新的地图。而地图控件负责把这些输出图片显示在同一个界面上，如图 4.17 所示。对于控件而言，每个资源就像一个图层。从资源可以到达数据源本身。

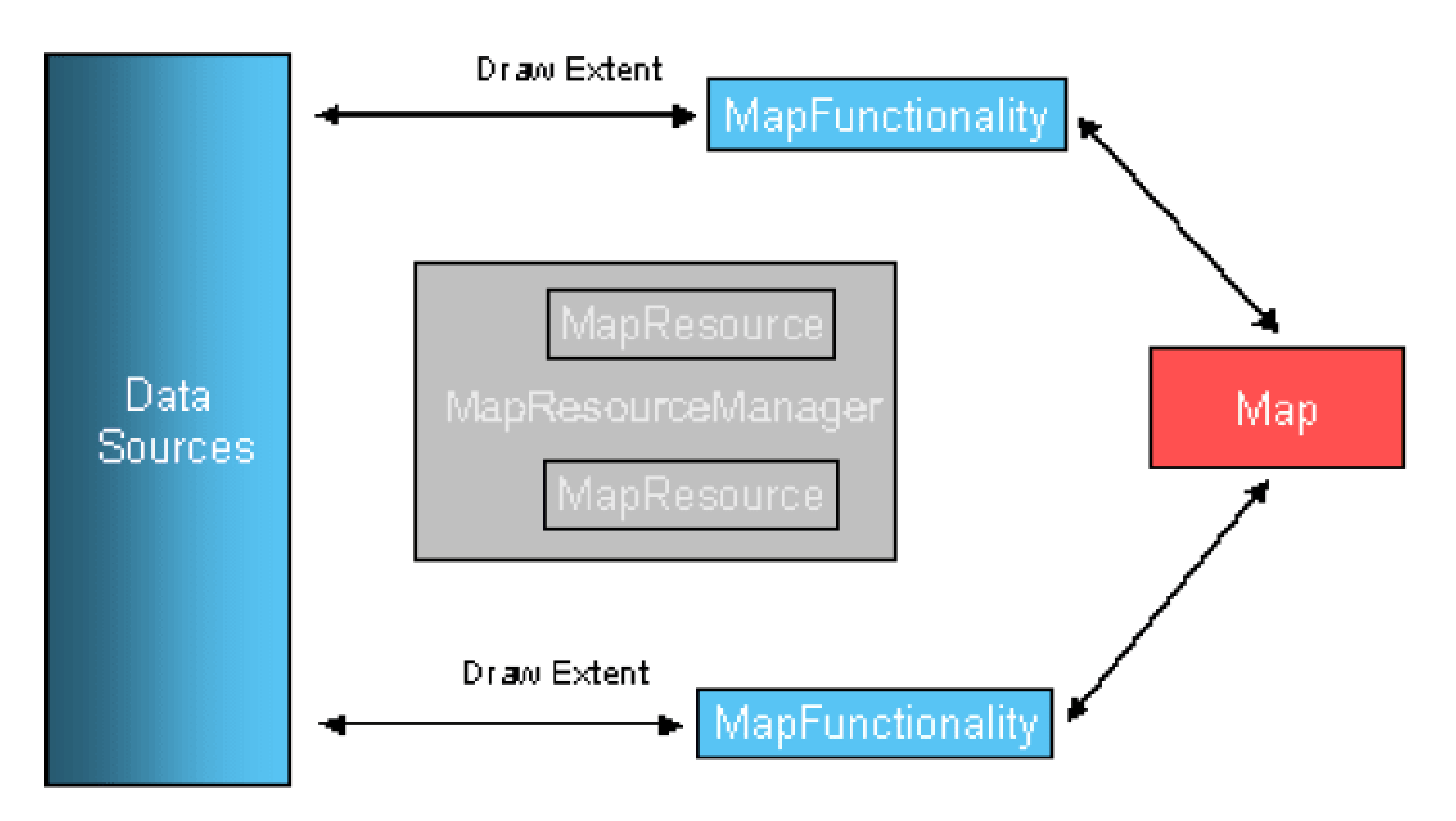


图 4.17　多个数据源的地图显示

上面这些文字可能刚开始看起来比较费劲，但是对于使用 ArcGIS Server 开发的人员来说一定要弄清楚这些关系。我们这里总结一下，可以将该多源 Web ADF 包括的类划分为三类，分别是：

- 自定义 ASP.NET Web 控件；
- 公有 API，包括公有数据源、资源与功能类；
- 数据源特有 API。

4.2.2　与 Web 应用程序框架相关的安装内容

Web ADF 安装主要包含三个部分，第一个是组件，第二个是与 Visual Studio 集成相关内容，第三个是一些帮助文档。

Web ADF 相关组件又包含不同部分，分别如下：

（1）Web ADF .NET 程序集

这些程序集位于 ArcGIS 安装路径的 DotNet 文件夹（默认安装路径为 C:\Program Files\ArcGIS\DotNet），主要内容如图 4.18 所示。包含 Web ADF 的控件、API 以及数据源的实现，还包含了 ArcIMS API 与 ArcGIS Server SOAP API。

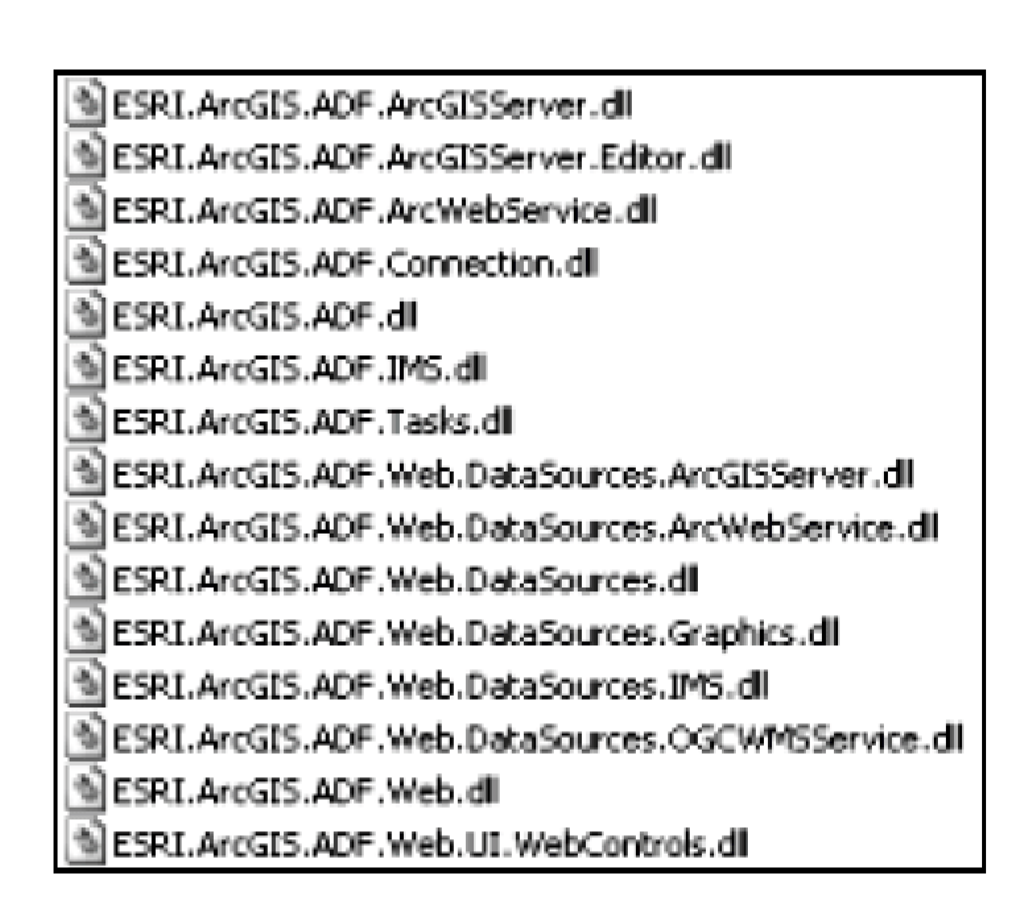


图 4.18　Web ADF .NET 程序集主要内容

（2）ArcObjects .NET 互操作程序集与 COM 对象库

由于 ArcObjects 是一系列的 COM 组件，因此需要一些.NET 互操作程序集才能正常访问这些组件。.NET 程序集位于 ArcGIS 安装路径的 DotNet 文件夹中，而 COM 对象库位于 ArcGIS 安装路径的 com 文件夹，内容如图 4.19 所示。

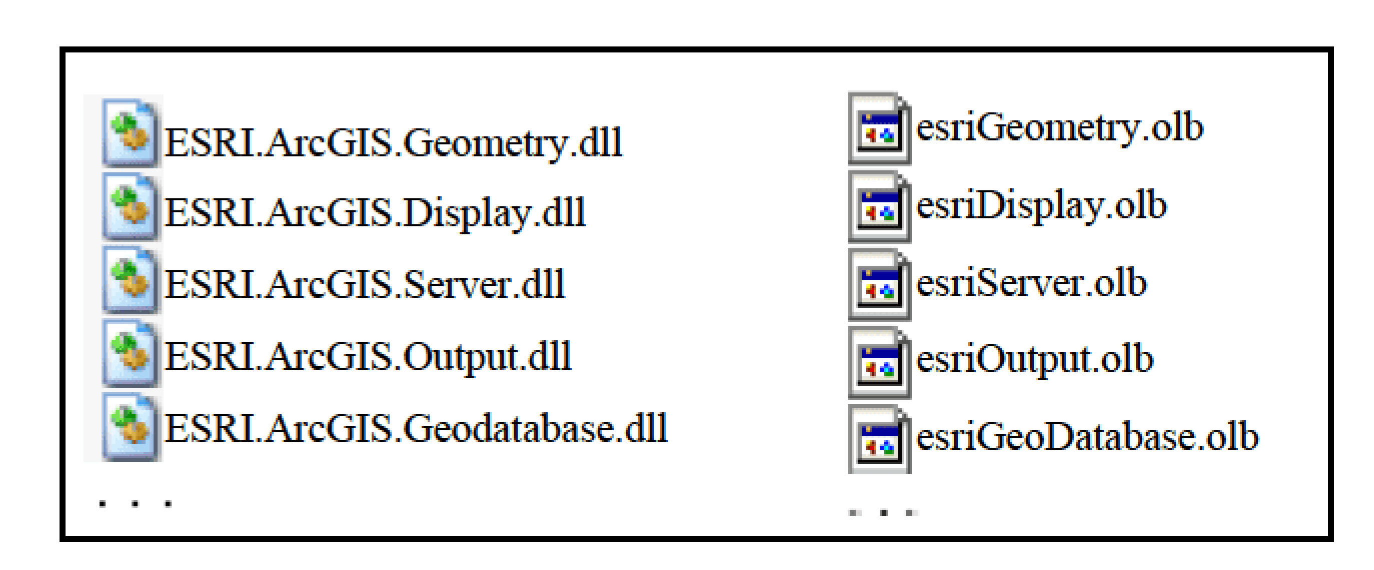


图 4.19　ArcObjects .NET 互操作程序集与 COM 对象库的主要内容

（3）JavaScript 库与客户端支持文件

这些文件用于支持客户端（即浏览器）使用 Web ADF 控件，包括 JavaScript 文件、图片以及样式表，位于 IIS 根路径（通常为 c:\inetpub\wwwroot）的 aspnet_client 文件夹中。多源（9.2）Web ADF 支持文件位于 ESRI 文件夹中，而单源（9.0\9.1）文件位于 esri_arcgis_server_webcontrols 文件夹中。aspnet_client 文件夹内容如图 4.20 所示。

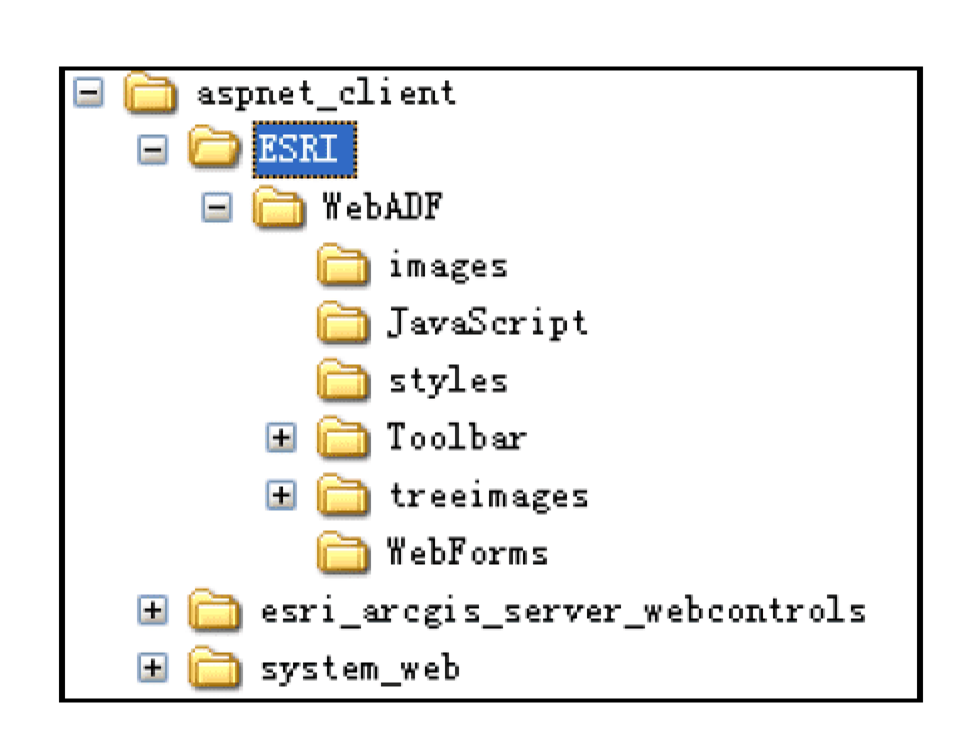


图 4.20　aspnet_client 文件夹的内容

与 Visual Studio 集成的内容主要包括位于工具箱中的一系列的 Web ADF 控件、Web 地图应用模板、集成的帮助系统以及一些属性设置对话框。

帮助文档集成在 Visual Studio 中的文档之中，除此之外，还包含有一些实例代码，位于 ArcGIS 安装路径的 DeveloperKit\SamplesNET\Server 文件夹中。

4.3　部分页面刷新的实现——Ajax

在 ASP.NET 网页的默认模型中，单击按钮或执行一些其他操作会导致回发，此时将重新创建页及其控件，并在服务器上运行页代码，且新版本的页被呈现到浏览器。但是，在有些情况下，需要从客户端运行服务器代码，而不执行回发。如果页中的客户端脚本维护一些状态信息（例如局部

变量值），那么发送页和获取页的新副本就会损坏该状态。此外，页回发会导致处理开销，这会降低性能，且会让用户不得不等待处理并重新创建页。

若要避免丢失客户端状态并且不导致服务器往返的处理开销，可以使用客户端回调。在客户端回调中，客户端脚本函数会向 ASP.NET 网页发送一个请求。该网页运行其正常生命周期的修改版本——初始化页并创建其控件和其他成员，然后调用特别标记的方法。该方法执行代码中编写的处理过程，然后向浏览器返回可由另一客户端脚本函数读取的值。在此过程中，该页一直驻留在浏览器中。

回调使用了一系列的标准技术，这些技术统称为 Ajax（Asynchronous JavaScript and XML，异步 JavaScript 和 XML）。

4.3.1　Ajax 技术

Ajax 本身并不是一项新技术，而是由 JavaScript、XML、XSLT、CSS、DOM 和 XMLHttpRequest 等多种技术组成的。下面是 Ajax 应用程序所用到的一些基本技术：

- HTML——用于建立 Web 表单并确定应用程序其他部分使用的字段;
- JavaScript——JavaScript 代码是运行 Ajax 应用程序的核心代码，帮助改进与服务器应用程序的通信;
- DHTML 或 Dynamic HTML——用于动态更新表单。我们将使用 div、span 和其他动态 HTML 元素来标记 HTML;
- 文档对象模型 DOM——用于（通过 JavaScript 代码）处理 HTML 结构和（某些情况下）服务器返回的 XML。

4.3.2　Ajax 及 XMLHttpRequest 对象原理

Ajax 的原理简单来说通过 XMLHttpRequest 对象来向服务器发异步请求，从服务器获得数据，然后用 Javascript 来操作 DOM 而更新页面。这其中最关键的一步就是从服务器获得请求数据。要清楚这个过程和原理，我们必须对 XMLHttpRequest 有所了解。

XMLHttpRequest 是 Ajax 的核心机制，它是在 IE5 中首先引入的，是一种支持异步请求的技术。简单的说，也就是 Javascript 可以及时向服务器提出请求和处理响应，而不阻塞用户，达到无刷新的效果。

所以我们先从 XMLHttpRequest 讲起，来看看它的工作原理。

首先，我们先来看看 XMLHttpRequest 这个对象的属性。它的属性有：

- onreadystatechange，每次状态改变所触发事件的事件处理程序;
- responseText，从服务器进程返回数据的字符串形式;
- responseXML，从服务器进程返回的 DOM 兼容的文档数据对象;
- status，从服务器返回的数字代码，比如常见的 404（未找到）和 200（已就绪）等;
- statusText，伴随状态码的字符串信息;

- readyState，对象状态值。各值所代表的意义如下：
  - 0，对象已建立，但是尚未初始化（尚未调用 open 方法）；
  - 1，对象已建立，尚未调用 send 方法;
  - 2，send 方法已调用，但是当前的状态及 http 头未知;
  - 3，已接收部分数据，因为响应及 http 头不全，这时通过 responseBody 和 responseText 获取部分数据会出现错误;
  - 4，数据接收完毕，此时可以通过通过 responseXML 和 responseText 获取完整的回应数据。

XMLHttpRequest 仅仅用来向服务器发出一个请求的，它的作用也局限于此，但它的作用是整个 Ajax 实现的关键，因为 Ajax 无非是两个过程，发出请求和响应请求。并且它完全是一种客户端的技术。而 XMLHttpRequest 正是处理了服务器端和客户端通信的问题所以才会如此的重要。

可以把服务器端看成一个数据接口，它返回的是一个纯文本流，当然，这个文本流可以是 XML 格式，可以是 Html，可以是 Javascript 代码，也可以只是一个字符串。这时候，XMLHttpRequest 向服务器端请求这个页面，服务器端将文本的结果写入页面，这和普通的 web 开发流程是一样的，不同的是，客户端在异步获取这个结果后，不是直接显示在页面，而是先由 Javascript 来处理，然后再显示在页面。至于现在流行的很多 Ajax 控件，比如 MagicaJax 等，可以返回 DataSet 等其他数据类型，只是将这个过程封装了的结果，本质上他们并没有什么太大的区别。

4.3.3 用 XMLHttpRequest 来实现 Ajax

下面我们通过一个实例来演示如何利用 XMLHttpRequest 来实现 Ajax。代码工程对应的文件夹名为 SimpleAjax。

新建一个 C#文件系统的 Web 站点，命名为 SimpleAjax。实例使用了 SQL Server Express 数据库，数据库中包含一个数据表，表名为 Items。数据表中包含的字段有 ItemID、ItemName 与 Quantity，分别代表货物标识、货物名称与仓储量。该数据库数据在工程的 App_Data 文件夹中。

在 Web.Config 文件中加入如下设置连接字符串代码：

```
<connectionStrings>
        <add name="Items"
              connectionString="Data Source=.\SQLEXPRESS;
                   AttachDbFilename=|DataDirectory|\Items.mdf;
                   Integrated ecurity=True;
                   User Instance=True"
              providerName="System.Data.SqlClient" />
</connectionStrings>
```

在默认的页面 Default.aspx 中加入一 SqlDataSource 控件，代码如下：

```
<asp:SqlDataSource ID="SqlDataSource1" runat="server"
   ConnectionString="<%$ ConnectionStrings:Items %>"
   SelectCommand="SELECT [ItemID], [ItemName] FROM [Items]">
</asp:SqlDataSource>
```

然后在 Default.aspx 中加入一个标签、一个列表框以及一个 div，代码如下：

```
<label for="ItemList" accesskey="I">
      货物名称：</label>
<asp:ListBox runat="server" ID="ItemList"
   DataSourceID="ItemsSource" DataTextField="ItemName"
   DataValueField="ItemID" EnableViewState="False"></asp:ListBox>
<div>
   <span id="ItemQuantityDisplay"></span>
   <span id="ProgressDisplay" style="display: none">
      <img src="images/loading.gif" alt="Loading" />
      检查仓储量...</span>
</div>
```

为了使用 JavaScript 来获取货物的仓储量，需要在 Web 站点提供一个 URL。最直接的方式是通过 HTTP 的 GET 协议来访问 ASP.NET 的 Web 服务。该方式的 URL 形式为：

```
http://hostname/path/<webservice>/<methodname>?<name1>=<value1>&<name2>=
<value2>
```

需要在 Web.Config 文件的<webServices>节中加入配置，才能使用 HTTP GET 方式访问 ASP.NET 的 Web 服务，配置代码如下：

```
<webServices>
  <protocols>
    <add name="HttpGet"/>
  </protocols>
</webServices>
```

下面需要我们创建一个 Web 服务来提供一个方法，用于获取一货物的仓储量。在当前站点中增加一个名为 WarehouseService 的 Web 服务，在其中增加一个 GetItemQuantity 方法，代码如下：

```
[WebMethod]
public int GetItemQuantity(string itemID) {
   if (String.IsNullOrEmpty(itemID))
      return 0;
   System.Threading.Thread.Sleep(500);
   using (SqlConnection conn = new SqlConnection(
     WebConfigurationManager.ConnectionStrings["Items"].ConnectionString))
   using (SqlCommand cmd = new SqlCommand(
     "SELECT [Quantity] FROM Items WHERE [ItemID] = @itemID", conn)) {
      conn.Open();
      cmd.Parameters.AddWithValue("@itemID", new Guid(itemID));
      return (int)cmd.ExecuteScalar();
   }
}
```

假设该 Web 服务位于本计算机上，想要获取仓储量的货物的 ID 为 1，那么可以使用如下的 URL 来实现：

http://localhost/WareHouseService.asmx/GetItemQuantity?itemID=1

上述 URL 将返回如下形式的响应：

```
<int xmlns="…">85</int>
```

余下来要完成的是使用 JavaScript 与服务器进行异步通讯。在我们这里主要是处理列表框 ItemList 控件的选择项改变事件。这些 JavaScript 代码可以嵌在页面文件中或放在另一个单独的文件中。由于通常这些脚本内容很多，因此通常宜放在单独文件中。

在当前站点中加入名为 ScriptLibrary 的文件夹，并在该文件夹中加入名为 SimpleAjaxExample.js 的 JavaScript 文件。该文件内容如下：

```
var itemList, itemQuantityDisplay, progressDisplay;
window.onload = function() {
    itemList = document.getElementById("ItemList");
    itemQuantityDisplay = document.getElementById("ItemQuantityDisplay");
    progressDisplay = document.getElementById("ProgressDisplay");

    if (!window.XMLHttpRequest) {
        window.XMLHttpRequest = function(){
            return new ActiveXObject("Microsoft.XMLHTTP");
        }
    }

    if (itemList.attachEvent) {
        itemList.attachEvent("onchange", itemList_OnChange);
    }
    else {
        itemList.addEventListener("change", itemList OnChange, false);
    }
}

function itemList OnChange() {
    if (!itemList.value){
        itemQuantityDisplay.style.display = "none";
    }

    var xr = new XMLHttpRequest();
    xr.open("GET","WarehouseService.asmx/GetItemQuantity?itemID="+ itemList.
value, true);
    xr.onreadystatechange = function() {
        if (xr.readyState == 4) {
            if (xr.status == 200) //Successful Request {
                var doc = xr.responseXML;
                var qty;

                if (doc.evaluate) //XML Parsing in Mozilla {
                    qty = doc.evaluate("//text()", doc,
                              null,
                              XPathResult.STRING_TYPE, null).stringValue;
                }
                else {
```

```
                //XML Parsing in IE
                qty = doc.selectSingleNode("//text()").data;
            }

            itemQuantityDisplay.innerHTML = "仓储量: " + qty;
            itemQuantityDisplay.className = "";
        }
        else {
            itemQuantityDisplay.innerHTML = "获取仓储量错误";
            itemQuantityDisplay.className = "Error";
        }

        itemQuantityDisplay.style.display = "";
        progressDisplay.style.display = "none";
    }
}
xr.send(null);
progressDisplay.style.display = "";
itemQuantityDisplay.style.display = "none";
}
```

XMLHttpRequest 对象在 Internet Explorer 7.0、Mozilla Firefox、Opera 与 Safari 中是一内嵌对象，在 Internet Explorer 6.0 中可作为 ActiveX 对象来获取。为处理这些不同情况，需要使用如下的 JavaScript 代码处理：

```
if (!window.XMLHttpRequest) {
   window.XMLHttpRequest = function() {
     return new ActiveXObject("Microsoft.XMLHTTP");
   }
}
```

该代码首先检查是否存在 XMLHttpRequest 对象，如果不存在，则在 window 对象上增加一个属性。

由于 Mozilla Firefox 与其他一些兼容 W3C DOM 的浏览器提供名为 addEventListener 方法来指定处理事件的函数，而 Internet Explorer 提供的方法是 attachEvent，因此我们为 ItemList 列表框控件的选择项改变事件增加事件处理器的代码如下：

```
if (itemList.attachEvent) {
  itemList.attachEvent("onchange", itemList_OnChange);
}
else {
  itemList.addEventListener("change", itemList_OnChange, false);
}
```

上述代码首先检查是否存在 attachEvent 方法，如果存在，则调用 attachEvent 方法，否则调用 addEventListener。

主要工作是在 itemList_OnChange 函数中完成的。在该函数中，可看到使用 XMLHttpRequest 对象的四个步骤：

（1）初始化 XMLHttpRequest 对象。

（2）调用 XMLHttpRequest 对象的 open 方法。该方法的第一个参数是向服务器提交数据的类型，即 POST 还是 GET。第二个参数是请求的 URL 地址。第三个参数是传输方式，false 为同步，true 为异步。默认为 true。如果是异步通信方式，客户机就不等待服务器的响应；如果是同步方式，客户机就要等到服务器返回消息后才去执行其他操作。

（3）提供处理 onreadystatechange 事件的处理函数。当 XMLHttpRequest 对象的状态改变时就会调用该处理函数。

（4）调用 send 方法。send 方法将请求的内容作为一字符串来发送。当使用 HTTP POST 方始时有用。

当 HTTP 请求发送时，不管是成功还是失败，XMLHttpRequest 对象的 readyState 设置为 4，表示发送完成。需要利用返回的 HTTP 状态代码来判断请求是否成功。状态值为 200 表示成功。一旦请求完成，可使用 responseXML（XML DOM 文档）或 responseText（普通文本）属性来得到 HTTP 响应。

我们这里处理的是 responseXML 属性。由于货物的仓储量保存在 XML 文档的根节点，因此 XPATH 的//text()是要提取的内容。IE 与 Mozilla Firefox 浏览器使用 XPath 提取方法有所不同。在 IE 中，可使用 selectSingleNode 函数，而在 Mozilla 中可使用 evaluate 函数。

运行应用程序。当用户从列表框中选择某货物后，可在页面看到一动画效果以及显示的“检查仓储量...”文字。随着动画消失，即可看到显示该货物的仓储量或是一错误信息。如图 4.21 所示。在过程中用户并没有看到页面闪烁，该过程即称为持续交互。

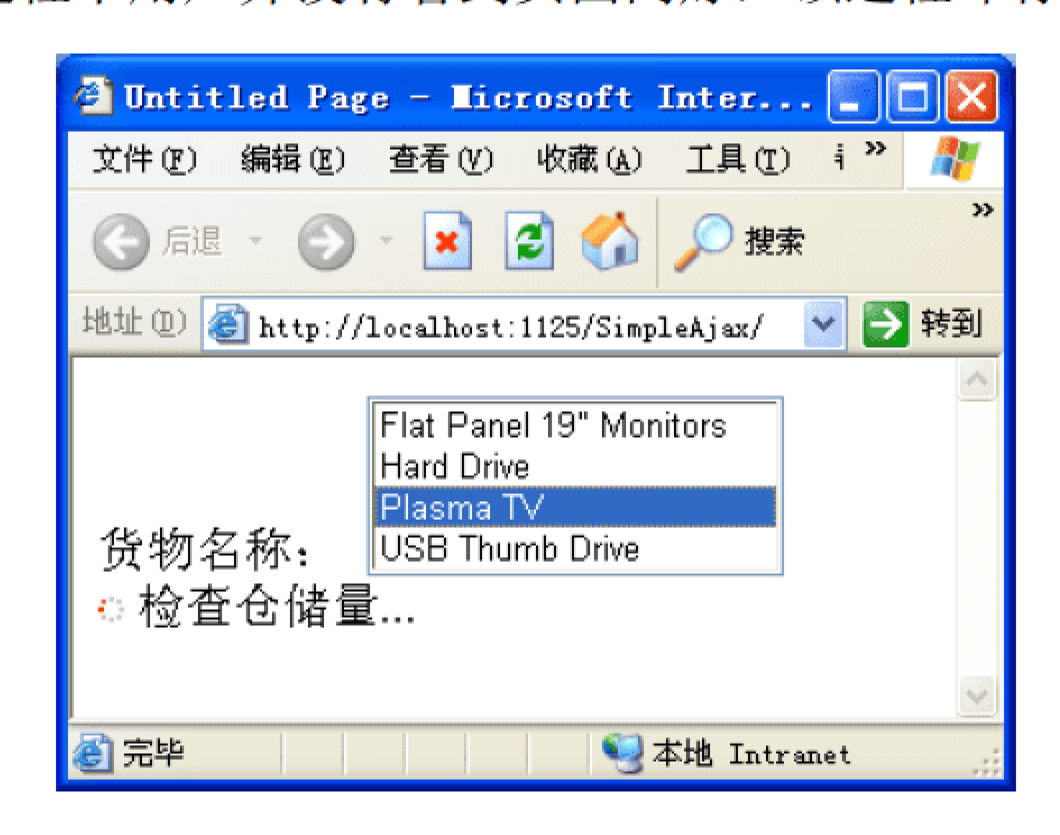




图 4.21 程序运行效果

这种持续交互是需要我们在客户端写大量的代码的代价换来的。幸运的是，这些客户端代码可以封装为平台中的通用代码来简化。ASP.NET 中的回调就提供了一个基础框架，实现在客户端调用服务器端的代码，也就是我们下一小节要介绍的内容。

4.3.4 .NET 中内置的 Ajax

在期望不执行回发而从客户端运行服务器代码的情况下，可以使用 ClientScriptManager 类来调用客户端回调。这称为对服务器执行带外回调。在客户端回调中，客户端脚本函数向 ASP.NET 网页发送异步请求。网页修改其正常生命周期来处理回调。

回调的过程如下所示：

（1）客户端发出请求，请求内容包括：指定更新控件和传递参数。
（2）服务端响应、处理。
（3）服务端将处理后的内容以字符串返回。
（4）客户端使用一个函数接受返回值。
（5）客户端更新指定控件。

使用回调的步骤如下：

（1）在控件或 Page 类中实现 ICallbackEventHandler 接口。该接口有两个方法，分别是 RaiseCallbackEvent 和 GetCallbackResult。RaiseCallbackEvent 方法是回调执行的方法，该方法处理回调的内容。它没有返回值，而是从浏览器接受一个字符串作为事件的参数，即该方法接受客户端 JavaScript 所传递的参数。注意它是首先触发的。接下来触发的就是 GetCallbackResult 方法，它将所得到的结果传回给客户端的脚本。

（2）生成调用该回调的客户端脚本。可通过调用 ClientScriptManager 类的 GetCallbackEventReference 方法来生成。Page 类的 ClientScript 属性就是一 ClientScriptManager 类的实例。

（3）编写代码调用在第二步骤中生成的客户端脚本。通常是在事件处理器中完成。

还是来实现上个例子的内容。首先在站点中加入一个名为 CallbackExample 的页面。在该类中加入如下代码，实现 ICallbackEventHandler 接口。

```
public partial class CallbackExample :
System.Web.UI.Page, ICallbackEventHandler {
    string callbackResult;

    #region ICallbackEventHandler Members

    public string GetCallbackResult() {
        return callbackResult;
    }

    public void RaiseCallbackEvent(string eventArgument) {
        try {
            callbackResult = Warehouse.GetItemQuantity(
               eventArgument).ToString();
        }
        catch (Exception e) {
            Trace.Write(e.Message);
            throw new Exception("检查仓储量失败");
        }
    }

    #endregion
}
```

RaiseCallbackEvent 方法的 eventArgument 参数从客户端脚本传过来。在例子中，就是 ItemList

列表框中的选项 ItemID。RaiseCallbackEvent 需要实现的就是得到由 eventArgument 指定的货物的仓储量，并把该值赋给页面中的一个字符串成员变量。我们需要一个成员来保存结果，并在 GetCallbackResult 方法中返回该变量。

下一步要实现的是生成调用的客户端脚本。可在 Page 的 Load 事件处理器中完成，代码如下：

```
protected void Page_Load(object sender, EventArgs e) {
    if (IsCallback)
        return;

    // 根据传入的参数返回实际的回调脚本
    string callBackFunctionCall =
                ClientScript.GetCallbackEventReference(
                this,
                "getSelectedItemID()",
                "onCallbackComplete",
                null,
                "onCallbackError",
                true
      );

    Page.ClientScript.RegisterClientScriptBlock(
        GetType(),
        "CallBack",
        "function DoClientCallBack() { " + callBackFunctionCall + "} ",
        true
    );
}
```

生成客户端脚本需要两个步骤。首先需要调用 GetCallbackEventReference，该方法包含六个参数：

（1）第一个参数是实现 ICallbackEventHandler 接口的类的实例，在例子中就是 Page 自身。

（2）第二个参数是从客户端脚本传递到服务器端的 RaiseCallbackEvent 方法的值。该参数应该是个得到一字符串的 JavaScript 表达式。在我们的例子中，需要传递的是列表框中选择的值。因此我们创建一个名为 getSelectedItemID 的函数，该函数返回在列表框中选择的值。

（3）第三个参数是一个客户端事件处理程序的名称，该处理程序接收服务器端 GetCallbackResult 返回的结果。

（4）第四个参数是与回调实例关联的上下文。如果有许多回调，那么可以用该参数来区分。该参数也应该是一 JavaScript 表达式。

（5）第五个参数是客户端处理回调失败的 JavaScript 函数的名称。

（6）最后一个参数指定回调是同步还是异步。由于我们希望执行异步操作，因此将该参数的值设置为 true。

当然 GetCallbackEventReference 方法有几个重载方法，如表 4-1 所示。

表 4-1　GetCallbackEventReference 方法重载列表

方法	说明
GetCallbackEventReference (Control, String, String, String)	获取一个对客户端函数的引用；调用该函数时，将启动一个对服务器端事件的客户端回调。此重载方法的客户端函数包含指定的控件、参数、客户端脚本和上下文。
GetCallbackEventReference (Control, String, String, String, Boolean)	获取一个对客户端函数的引用；调用该函数时，将启动一个对服务器端事件的客户端回调。此重载方法的客户端函数包含指定的控件、参数、客户端脚本、上下文和布尔值。
GetCallbackEventReference (Control, String, String, String, String, Boolean)	获取一个对客户端函数的引用；调用该函数时，将启动一个对服务器端事件的客户端回调。此重载方法的客户端函数包含指定的控件、参数、客户端脚本、上下文、错误处理程序和布尔值。
GetCallbackEventReference (String, String, String, String, String, Boolean)	获取一个对客户端函数的引用；调用该函数时，将启动一个对服务器端事件的客户端回调。此重载方法的客户端函数包含指定的目标、参数、客户端脚本、上下文、错误处理程序和布尔值。 我们就整个程序作个系统的说明，并且列出前台的页面代码和后台的逻辑代码，这样可以使得你对程序有个直观的理解。

从 GetCallbackEventReference 方法返回的值是一异步调用服务器的 JavaScript 表达式。将该值放在另一个名为 DoClientCallBack 的 JavaScript 函数中，然后通过列表框的选项改变事件处理器调用该函数。使用 ClientScript.RegisterClientScriptBlock 方法来组织该 JavaScript 函数并注册。这样我们就完成了服务器端的工作，下面要完成的是客户端代码。

客户端代码如下：

```
var itemList, itemQuantityDisplay, progressDisplay;

window.onload = function() {
    itemList = document.getElementById("ItemList");
    itemQuantityDisplay = document.getElementById("ItemQuantityDisplay");
    progressDisplay = document.getElementById("ProgressDisplay");

    if (itemList.attachEvent) {
        itemList.attachEvent("onchange", itemList OnChange);
    }
    else {
        itemList.addEventListener("change", itemList_OnChange, false);
    }
}

function itemList OnChange() {
    document.getElementById("ProgressDisplay").style.display = "";
    document.getElementById("ItemQuantityDisplay").style.display = "none";
    DoClientCallBack();
}
```

```
// 回调成功
function onCallbackComplete(result, context) {
  progressDisplay.style.display = "none";

  itemQuantityDisplay.className = "";
  itemQuantityDisplay.style.display = "";
  itemQuantityDisplay.innerHTML = result + " in stock";
}

// 回调错误
function onCallbackError() {
  progressDisplay.style.display = "none";

  itemQuantityDisplay.className = "Error";
  itemQuantityDisplay.style.display = "";
  itemQuantityDisplay.innerHTML = "Error retrieving quantity";
}

function getSelectedItemID() {
    return itemList.value;
}
```

该页面的用户交互与前面一小节的页面是一样的，但是代码大大减少了，因为我们不需要编写代码来解析 XML 或处理 XMLHttpRequest 对象。

4.3.5 ArcGIS Web 应用开发框架中的 Ajax

ESRI ArcGIS Server 9.2 的 Web 应用开发框架充分利用了 ASP.NET 中的回调技术。我们来看看利用 Visual Studio 模板创建的默认应用程序中如何利用回调的。

首先看到默认的主页面 Default.aspx 对应的类 WebMapApplication 实现了 ICallbackEventHandler 接口。

我们再以 Identify 工具来深入了解应用开发框架。

在 WebMapApplication 类的 Load 事件处理器中，调用 new MapIdentify(Map1)初始化 Identify 工具。在 MapIdentify 类的构造函数中，调用了 SetupIdentify 方法。该方法首先通过调用 GetCallbackEventReference 方法生成客户端脚本，然后调用 RegisterClientScriptBlock 注册一个名为 identifyCallbackFunctionString 的 JavaScript 函数，该函数中调用生成的客户端脚本。

在 display_mapidentify.js 文件的 MapIdClick 函数（Identify 事件处理器）中调用了 identifyCallbackFunctionString 函数，从而触发了 WebMapApplication 类的 RaiseCallbackEvent 方法。

RaiseCallbackEvent 方法不仅需要处理 Identify 工具，还需要处理其他工具，本实例中包括关闭应用程序与获取版权文本。为了在该方法中判断是哪个工具启动了该方法调用，需要在传入在参数中包含表明调用“身份”。同时由于在传入参数中还需要其他可能更多的信息，例如 Identify 工具，不仅需要表明调用者身份的信息，还需要当前用户单击处 X、Y 坐标信息。为了能在

RaiseCallbackEvent 方法中把这些信息都解释出来，本实例使用的是

“参数名=参数值&参数名=参数值”

式。例如在 MapIdClick 函数中传入的是

"ControlID=Map1&ControlType=Map&EventArg=MapIdentify&Map1_mode=MapIdentify&minx=" + zleft + "&miny=" + ztop。

而在 RaiseCallbackEvent 方法中，通过调用 Split("&".ToCharArray())方法将各个“参数名=参数值”放置到一组数列中，然后再通过下面的代码将“参数名”与“参数值”分开：

```
if (keyValuePairs.Length > 0) {
    for (int i = 0; i < keyValuePairs.Length; i++) {
        keyValue = keyValuePairs.GetValue(i).ToString().Split(
           "=".ToCharArray());
        m queryString.Add(keyValue[0], keyValue[1]);
    }
}
else {
    keyValue = responseString.Split("=".ToCharArray());
    if (keyValue.Length > 0)
        m queryString.Add(keyValue[0], keyValue[1]);
}
```

通过执行上述的代码后，只需要调用 m_queryString["EventArg"]就能判断是哪个工具启动该方法的调用。如果是 Identify 工具，那么该值为 MapIdentify。

在判断出调用工具是 Identify 后，RaiseCallbackEvent 方法调用 MapIdentify 类的 Identify 方法得到返回值。

4.4　自定义工具与命令

前面所介绍的开发，不管是使用 Web Mapping Application 模板还是直接使用 Web 控件，使用的都是 Web ADF 中封装好的工具与命令，例如地图放大、缩小、漫游。然而在实际项目应用中，需要的功能远远不仅如此，至少还需要通过图形查询属性（点查询、矩形查询、多边形查询等）或是通过属性查询图形。因此，如何在工具条中增加新的自定义工具是 ArcGIS Server 开发者面临的第一个问题。

下面我们通过开发图形查询属性功能来介绍如何自定义工具与命令。按照 4.1.3 节介绍的方法创建一个同样的 Web 应用程序，将其工程名称命名为 CustomTool。

为了使实例更接近实际的应用，也为了介绍如何替换 Web ADF 自身附带的图标与样式文件，书中使用了自己设计的一套图标与样式文件。读者可以从本书随附的源代码文件（该文件可从 www.booksaga.com 网站下载）的 CustomTool 工程中找到相应的资源，并拷贝到对应的目录中。例如图标文件请拷贝到工程的 Images\Toolbar 子文件夹中，样式文件拷贝到工程的 css 子文件夹中。

4.4.1 在工具栏中增加按钮

要增加工具或命令，第一步当然是要在界面的工具栏中增加一个按钮。

由于书中提供的图标中每个都带有描述，因此不再需要描述信息。首先需要将工具栏控件 Toolbar1 的 ToolbarStyle 的属性由 ImageAndText 设置 ImageOnly，表示在工具栏中只显示工具按钮指定的图标，而不需要文字描述。

通过属性设置面板打开 Toolbar1 控件的 ToolbarItems 属性设置对话框 ToolbarCollectionEditorForm，并通过 Show Properties 按钮展开对话框的属性设置部分，如图 4.22 所示。

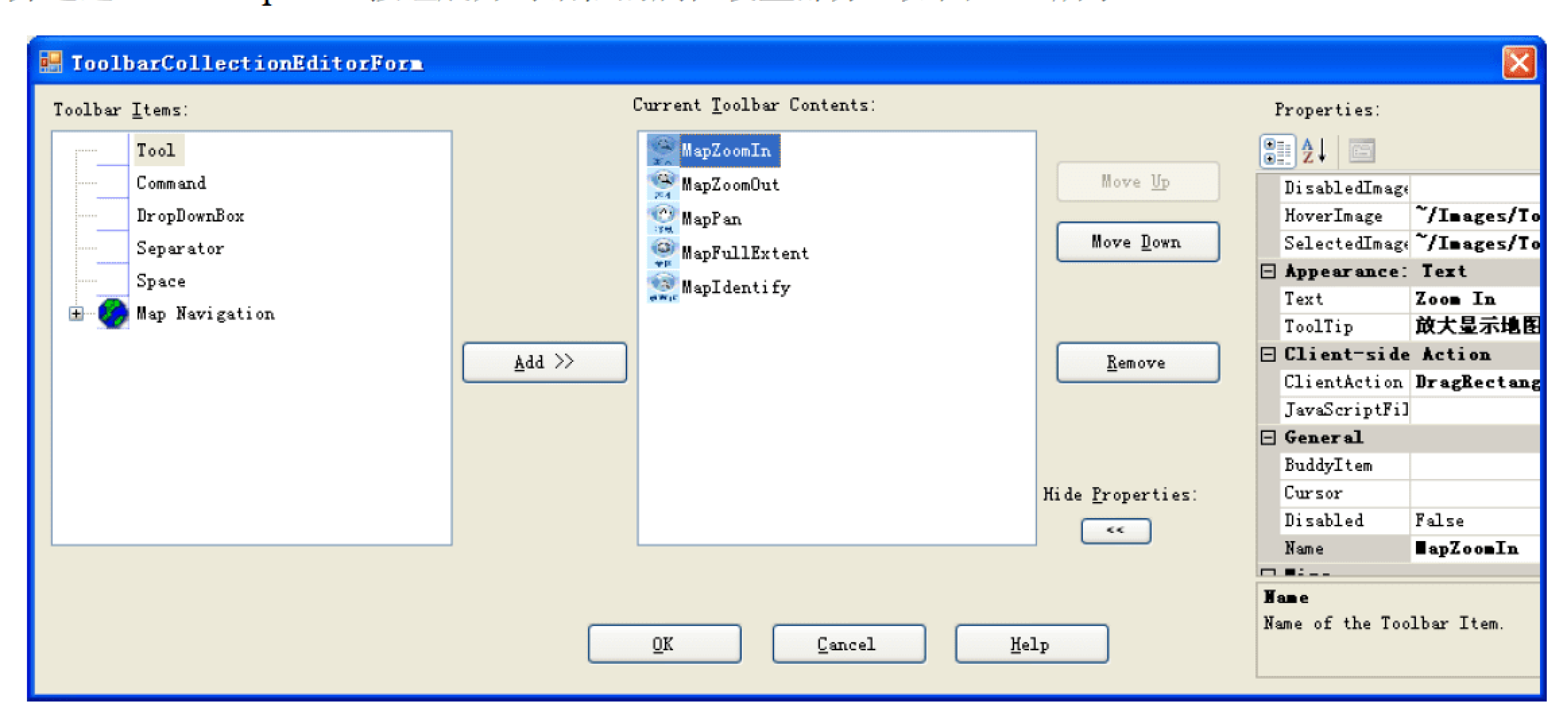


图 4.22 通过 ToolbarCollectionEditorForm 对话框控制工具栏中工具或命令按钮

首先在对话框的左边窗口中选择 Tool，然后选择 Add 按钮便可增加一个工具按钮，将其命名为 MapIdentify。

通过对话框右部的属性设置窗口，将原来几个 Web ADF 提供的工具按钮以及新增加的工具按钮的 DefaultImage、HoverImage 与 SelectedImage 属性设置为我们对应的图标所在目录的相对路径。

另一个更直接但需要一些经验的方式是直接修改 Default.aspx 中 <esri:Toolbar> 与 </esri:Toolbar>之间的代码。修改后的代码如下：

```
<esri:Toolbar ID="Toolbar1" runat="server"
    BuddyControlType="Map" Group="Toolbar1 Group"
    Height="50px" Width="198px"
    WebResourceLocation="/aspnet_client/ESRI/WebADF/"
    ToolbarStyle="ImageOnly">

    <ToolbarItems>
        <esri:Tool ClientAction="DragRectangle" Name="MapZoomIn"
           DefaultImage="~/Images/Toolbar/zoomin 1.jpg"
           HoverImage="~/Images/Toolbar/zoomin 2.jpg"
           SelectedImage="~/Images/Toolbar/zoomin 3.jpg"
           ServerActionAssembly="ESRI.ArcGIS.ADF.Web.UI.WebControls"
           ServerActionClass="ESRI.ArcGIS.ADF.Web.UI.WebControls.Tools.MapZoomIn"
```

```
            Text="Zoom In" ToolTip="放大显示地图" />
          <esri:Tool ClientAction="DragRectangle" Name="MapZoomOut"
             DefaultImage="~/Images/Toolbar/zoomout 1.jpg"
             HoverImage="~/Images/Toolbar/zoomout 2.jpg"
             SelectedImage="~/Images/Toolbar/zoomout 3.jpg"
             ServerActionAssembly="ESRI.ArcGIS.ADF.Web.UI.WebControls"
             ServerActionClass="ESRI.ArcGIS.ADF.Web.UI.WebControls.Tools.MapZ
oomOut"
             Text="Zoom Out" ToolTip="缩小显示地图" />
          <esri:Tool ClientAction="DragImage" Name="MapPan"
             DefaultImage="~/Images/Toolbar/pan 1.jpg"
             HoverImage="~/Images/Toolbar/pan 2.jpg"
             SelectedImage="~/Images/Toolbar/pan 3.jpg"
             ServerActionAssembly="ESRI.ArcGIS.ADF.Web.UI.WebControls"
             ServerActionClass="ESRI.ArcGIS.ADF.Web.UI.WebControls.Tools.MapP
an"
             Text="Pan" ToolTip="漫游" />
          <esri:Command ClientAction="" Name="MapFullExtent"
             DefaultImage="~/Images/Toolbar/fullextent 1.jpg"
             HoverImage="~/Images/Toolbar/fullextent 2.jpg"
             SelectedImage="~/Images/Toolbar/fullextent 3.jpg"
             ServerActionAssembly="ESRI.ArcGIS.ADF.Web.UI.WebControls"
             ServerActionClass="ESRI.ArcGIS.ADF.Web.UI.WebControls.Tools.MapF
ullExtent"
             Text="Full Extent" ToolTip="全图显示" />
          <esri:Tool ClientAction="Point" Name="MapIdentify"
             DefaultImage="~/Images/Toolbar/identify 1.jpg"
             HoverImage="~/Images/Toolbar/identify 2.jpg"
             SelectedImage="~/Images/Toolbar/identify 3.jpg"
             ServerActionAssembly="App Code"
             ServerActionClass="IdentifyPoint"
             Text="Identify" ToolTip="点查询" />
       </ToolbarItems>
       <BuddyControls>
          <esri:BuddyControl Name="Map1" />
       </BuddyControls>

    </esri:Toolbar>
```

从上面的代码我们可以看出在工具栏中有两类按钮，一类称为 Tool（工具），另一类为 Command（命令）。它们之间的区别在于，工具需要用户与地图交互，例如放大工具需要用户在地图上用鼠标拖出一个矩形框后才能执行放大操作，而命令则不要用户与地图交互，可直接执行操作，例如全图显示命令不需要用户在地图上有任何操作，而直接执行显示整个范围地图的操作。

在工具与命令按钮的属性中，DefaultImage、HoverImage 与 SelectedImage 分别表示正常状态、鼠标移动到该按钮上的状态以及选择状态时所使用的图标所在的相对于 Default.aspx 文件的相对路径。

ClientAction 属性表示该工具需要用户与地图交互的类型，可选的值有 Point、Line、PolyLine、Polygon、Circle、Oval、DragRectangle 与 DragImage。

ServerActionAssembly 属性表示当用户选择该了工具或命令，并且与地图交互完毕后（命令按钮不需要用户与地图交互），服务器端执行的代码所在的程序集的名称。而 ServerActionClass 属性表示该代码所在的类的名称。例如上面的放大、缩小、漫游与全图显示都执行的代码都在 ESRI.ArcGIS.ADF.Web.UI.WebControls 程序集中，分别对应 MapZoomIn、MapZoomOut、MapPan 与 MapFullExtent 类。

对于自定义的工具，也需要使用 ServerActionAssembly 与 ServerActionClass 属性指出服务器端代码所在位置。在上面的代码中指定的是 App_Code 文件夹中的 IdentifyPoint 类。

实现下面几个工具要编写的就是该类的代码。

4.4.2 自定义点查询工具

该工具要实现的是用户在地图上用鼠标通过单击选择某地物要素后，弹出一网页显示选择要素的属性，并高亮显示被选择的要素。为了简单，我们先只介绍如何通过点查询出要素的属性，在下一小节再设置高亮显示。

首先通过工程的右键菜单的 Add ASP.NET Folder 的 App_Code 命令，在当前工程中增加 App_Code 文件夹。ASP.NET 2.0 使用该文件夹存放应用程序级别的类库。

然后通过右键菜单的 Add New Item 命令，在 App_Code 文件夹中加入一个名为 IdentifyPoint.cs 的文件。

在 IdentifyPoint 类中用到了地图控件、点对象等，因此为了避免使用全路径来调用类，需要在 IdentifyPoint 类的前面加入如下命名空间的引用：

```
using ESRI.ArcGIS.ADF.Web;
using ESRI.ArcGIS.ADF.Web.Geometry;
using ESRI.ArcGIS.ADF.Web.DataSources;
using ESRI.ArcGIS.ADF.Web.UI.WebControls;
using ESRI.ArcGIS.ADF.Web.UI.WebControls.Tools;
```

自定义工具类必须实现 IMapServerToolAction 接口，需要增加两项代码。一个是在类声明的后面加入 IMapServerToolAction 接口，另一个是需要实现 IMapServerToolAction 接口的 ServerAction 方法（Visual Studio 可以在增加接口名称时自动插入该方法）。因此该类代码的框架如下：

```
public class GraphicPointTool: IMapServerToolAction
{
    void IMapServerToolAction.ServerAction(ToolEventArgs args)
    {
    }
}
```

现在要实现的是在 ServerAction 方法中增加代码，实现查询用户单击处要素的属性。该方法的 args 参数包含了用户单击处的屏幕坐标信息，从该参数还可以得到地图控件的引用。在 ServerAction 方法中先增加如下代码：

```
// 从方法的参数中得到地图控件的引用
Map map = args.Control as Map;
```

```
// 从方法的参数中得到用户单击位置的点
PointEventArgs pea = (PointEventArgs)args;
System.Drawing.Point screen_point = pea.ScreenPoint;
```

需要将屏幕像素坐标转换到地图坐标，代码如下：

```
Point point = Point.ToMapPoint(screen_point.X, screen_point.Y,
    map.Extent, (int)map.Width.Value, (int)map.Height.Value);
```

下面需要实现的是利用该点查询要素。要实现查询，则需要判断资源是否具有查询功能，代码如下：

```
IGISFunctionality gisfunc = map.GetFunctionality("NorthAmericaMap");
if (gisfunc == null)
    return;

IGISResource gisresource = gisfunc.Resource;
bool supportquery = gisresource.SupportsFunctionality(
                     typeof(IQueryFunctionality));
if (!supportquery)
    return;
```

在上面的代码中首先用“NorthAmericaMap”（地图资源管理器中包含要查询图层的资源的名称）作为参数名，调用地图对象的 GetFunctionality 方法，得到该资源能提供的功能，如果该功能对象不为空，即表示能提供查询、地图显示等功能，则接着执行，否则直接返回。然后调用资源对象的 SupportsFunctionality 方法判断是否具有查询功能，如果没有查询，则直接返回。如果有，则调用资源对象的 CreateFunctionality 方法得到查询功能对象，代码如下：

```
IQueryFunctionality qfunc;
qfunc = gisresource.CreateFunctionality(
   typeof(IQueryFunctionality), null) as IQueryFunctionality;
```

得到查询功能对象后，调用其 Identify 方法执行查询，代码如下：

```
System.Data.DataTable[] qdatatable = qfunc.Identify(
   null, point, 1, IdentifyOption.AllLayers, null);
```

查询功能 Identify 方法的第一个参数是地图功能名称；第二个参数是查询要素的图形对象；第三个参数是误差范围；第四个参数是查询方式，可以是 AllLayers、VisibleLayers 与 TopMostLayer 三个值之一，分别表示查询所有图层、查询可见图层与查询最上层图层；第五个参数是可查询图层的 ID，由于我们需要对所有图层查询，所以该值可使用 null。

至此点对象查询要素完成，下面要实现的是以网页的方式展现查询到的要素的属性。代码如下：

```
if (qdatatable == null)
    return;

System.Data.DataSet dataset = new System.Data.DataSet();
for (int i = 0; i < qdatatable.Length; i++) {
    dataset.Tables.Add(qdatatable[i]);
}
DataTableCollection dtc = dataset.Tables;
```

```
IdentifyHelper.ShowIdentifyResult(map, dtc);
```

在上述代码中，首先判断 qdatatable 是否为空，如果为空，直接返回。如果不为空，则将数组形式的 qdatatable 转换为集合形式的 dtc 变量。最后为了简化代码，也为了代码重用（其他查询工具也会利用），将显示查询结果的代码放到了 IdentifyHelper 类的 ShowIdentifyResult 方法中。该方法的代码如下：

```
public static void ShowIdentifyResult(Map map, DataTableCollection dtc)
{
    string returnstring = string.Empty;
    foreach (DataTable dt in dtc)  {
        if (dt.Rows.Count == 0)
            continue;
        returnstring += GetHtmlFromDataTable(dt);
    }

    returnstring = returnstring.Replace("\r\n", "");
    returnstring = returnstring.Replace("\n", "");
    string functionValue = "var theForm = document.forms[0];";
    functionValue += "theForm.FunctionValue.Value='" + returnstring + "';";
    functionValue += "open('IdentifyResult.htm', 'IdentifyResult');";
    AddJavaScriptCallback(map, functionValue);
}
```

上述代码调用 GetHtmlFromDataTable 方法，实现从数据表转换为 HTML 表格格式的字符串。在得到该字符串后，我们构造了一段 JavaScript 代码，该段 JavaScript 代码的意思是将该字符串的值赋给 Default.aspx 页面中的一个不可见文本框 FunctionValue，然后调用 open 方法弹出 IdentifyResult.htm 网页。最后调用 IdentifyHelper 类的 AddJavaScriptCallback 方法将该段 JavaScript 代码加入到地图对象的 CallbackResults 属性中。该段代码将最终在客户端（即浏览器）执行。

由于上面代码需要利用文本框 FunctionValue，因此需要在 Default.aspx 文件的</form>代码行之前加入如下内容：

```
<input type="hidden" name="FunctionValue" value="" />
```

在 App_Code 中加入名为 IdentifyHelper 的类。该类的 GetHtmlFromDataTable 方法代码如下：

```
public static string GetHtmlFromDataTable(DataTable dt) {
    GridView gd = new GridView();
    gd.ToolTip = dt.TableName;
    gd.Caption = dt.TableName;
    gd.DataSource = dt;
    gd.DataBind();
    gd.Visible = true;
    gd.BorderWidth = 0;
    gd.CssClass = "list-line";
    gd.CellPadding = 3;
```

```
        gd.CellSpacing = 1;
        gd.HeaderStyle.CssClass = "barbg";
        gd.HeaderStyle.HorizontalAlign = HorizontalAlign.Center;
        gd.RowStyle.CssClass = "listbg";

        string returnString = string.Empty;
        using (System.IO.StringWriter sw = new System.IO.StringWriter()) {
            HtmlTextWriter htw = new HtmlTextWriter(sw);
            gd.RenderControl(htw);
            htw.Flush();
            string tempStr = sw.ToString();
            returnString += tempStr;
        }
        return returnString;
    }
```

上述代码首先利用 GridView 将数据表的内容显示到表格中，其中 list-line、barbg 与 listbg 是为了美观而应用的样式。然后利用 HtmlTextWriter 类将表格的内容转换为 HTML 文本格式的字符串。

IdentifyHelper 类的 AddJavaScriptCallback 方法代码如下：

```
    public static void AddJavaScriptCallback(Map map, string executeString)
    {
        object[] oa = new object[1];
        oa[0] = executeString;
        CallbackResult cr = new CallbackResult(null, null, "javascript", oa);
        map.CallbackResults.Add(cr);
    }
```

这段代码最关键的类是 CallbackResult，它简化了 Web ADF 中客户端回调的处理，不用再创建自己的客户端和服务器端逻辑，使用 CallbackResult 就可以将信息传回客户端，更新客户端页面的内容、图片或执行 JavaScript 脚本。

看到这，读者可能仍然不明白为什么只需要通过上述代码，而不需要在客户端写代码，客户端就会执行我们在 IdentifyPoint 类 ServerAction 方法中编写的 JavaScript 代码。原因是 Web ADF 已经写好了客户端代码。所有工具按钮执行完服务器端代码后，客户端执行 C:\Inetpub\wwwroot\aspnet_client\ESRI\WebADF\JavaScript\display_dotnetadf.js 文件中的 processCallbackResult 方法，该方法中有如下一段代码：

```
    else if (action=="javascript") {
        validResponse = true;
        eval(actions[3]);
    }
```

该段代码判断如果服务器段的 CallbackResult 构造函数中第三个参数的内容是“javascript”，则执行第四个参数指定的代码。

当然在构造 CallbackResult 类的实例时，第三个参数除了为 javascript 外，还可以为 content、innercontent 与 image，各自含义可参考如下代码：

```
    if (action=="content") {
```

```
        validResponse = true;
        o = document.getElementById(actions[1]);
        if (o != null) {
            o.outerHTML=actions[3];
        }
    }
    else if (action=="innercontent") {
        validResponse = true;
        o = document.getElementById(actions[1]);
        if (o != null) {
            o.innerHTML=actions[3];
        }
    }
    else if (action=="image") {
        validResponse = true;
        o = document.images[actions[1]];
        if (o != null) {
            o.src = actions[3];
        }
        else alert (actions[1] + " was null");
    }
    else if (action=="javascript") {
        validResponse = true;
        eval(actions[3]);
    }
    else if(response.length>0) {
        alert(response + "\nContext: " + context);
    }
```

最后要完成的是利用网页将 FunctionValue 文本框中的内容显示出来。在工程中加入一个 HTML 文件，命名为 IdentifyResult.htm。其代码很简单，如下所示：

```
    <head>
        <title>图形查询结果</title>
        <link href="css/style1.css" type="text/css" rel="stylesheet"/>
        <script type="text/javascript" language="javascript">
            function loadResult() {
               var identifyResult = opener.document.forms[0].FunctionValue.Value;
               var o = document.getElementById("datadiv");
                 if (o != null) {
                    o.innerHTML=identifyResult;
                 }
            }
        </script>
    </head>
    <body>
        <div runat="server" id="datadiv">
        </div>
    </body>
    </html>
```

```
<script type="text/javascript" language="javascript">
    function window.onload() {
        loadResult();

         window.focus();
    }
</script>
```

至此我们就完成了点查询要素并显示其属性。运行应用程序，选择点查询，然后在地图上单击某个行政区划，系统就会弹出一个页面显示该行政区划的相关属性，例如图 4.23 显示的是加拿大萨斯喀彻温省的面积、人口等属性。

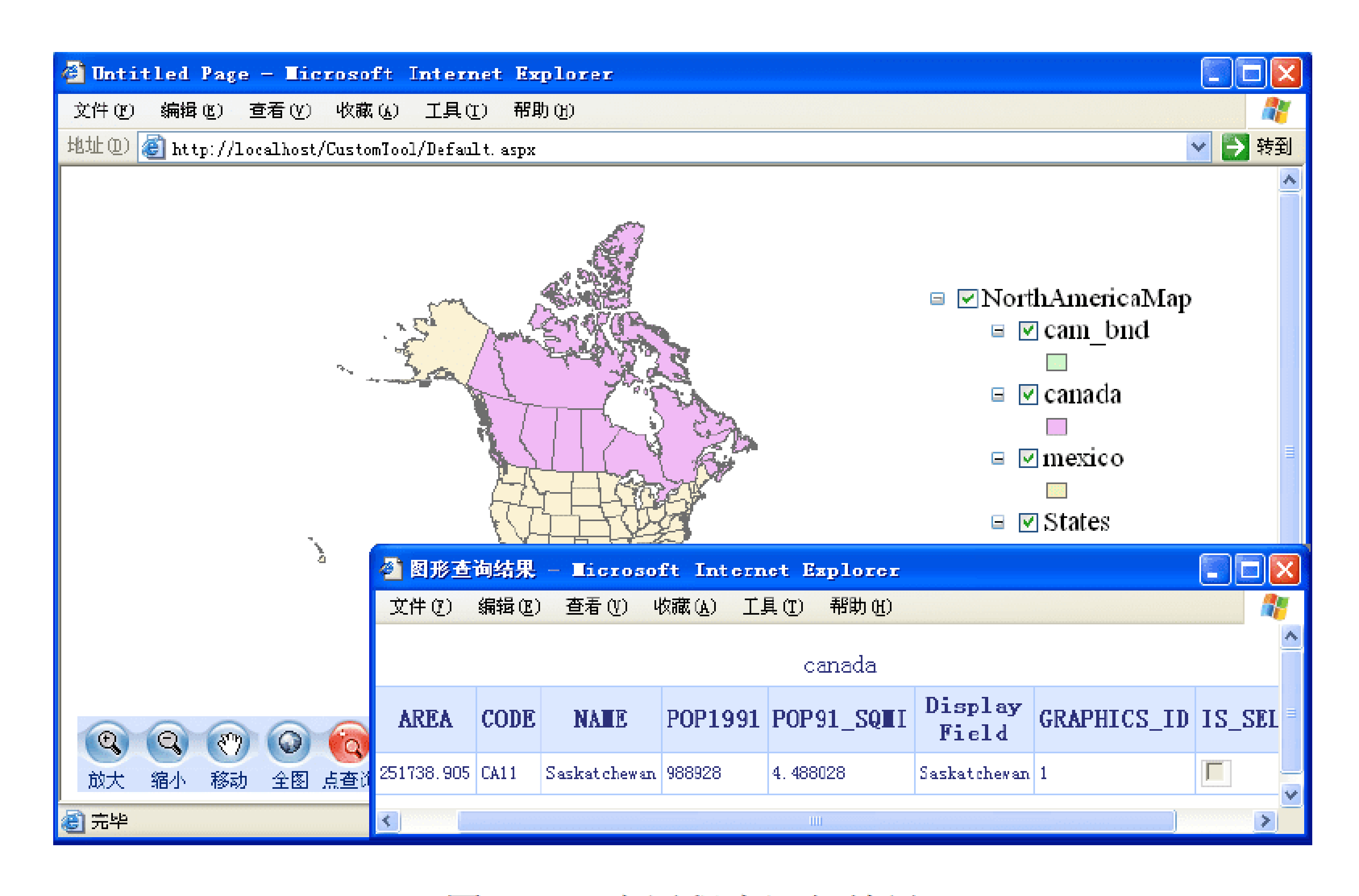


图 4.23　应用程序运行结果

4.4.3　高亮显示要素

在利用图形查询属性时，用户通常希望能同时看到有哪些要素被选择了。这个需求可以通过高亮显示要素来满足。

在 4.4.2 节中，我们已经查询出被选择的要素，下面要完成的就是给这些要素用另一种方式画出来。

在 IdentifyHelper 类中，加入如下所示的静态方法：

```
public static void HighLightShow(Map map, DataTableCollection dtc)
{
}
```

高亮显示要素一般可以通过两种方式实。一是在地图资源管理器中增加一 GraphicsLayer，然后在该图层中将要素重新画一次，另一种方式是设置资源绘图功能的 MapDescription 属性的 CustomGraphics 属性。我们在该节中介绍第二种方式。

当使用 ArcGIS Server 绘图功能时，MapDescription 属性提供了一个值对象，通过该对象可以修改由 ArcGIS Server 地图服务生成的地图的显示与内容。

在 HighLightShow 方法中首先要得到的当然是 NorthAmericaMap 资源的绘图功能，使用的代码如下：

```
MapFunctionality mf =
   (MapFunctionality)map.GetFunctionality("NorthAmericaMap");
```

通过绘图功能的 MapDescription 属性来访问 MapDescription 对象，代码如下：

```
MapDescription mapDescription = mf.MapDescription;
```

然后需要将原来高亮显示的要素去掉高亮，将 CustomGraphics 设置为 null 即可，代码如下：

```
mapDescription.CustomGraphics = null;
```

由于 MapDescription 类属于 ESRI.ArcGIS.ADF.ArcGISServer 程序集，此外还有其他对象需要利用 ESRI.ArcGIS.ADF.Web.DataSources.ArcGISServer 程序集，因此需要加入如下引用命名空间的代码：

```
using ESRI.ArcGIS.ADF.ArcGISServer;
using ESRI.ArcGIS.ADF.Web.DataSources.ArcGISServer;
```

此时编译代码则会报告 ArcGISServer 命名空间不存在，原因是我们没有在工程中加入上述两个程序集的引用。利用工程的右键菜单的 Add ArcGIS Reference 命令，打开如图 4.24 所示的对话框。在该对话框中分别选择 ESRI.ArcGIS.ADF.ArcGISServer 与 ESRI.ArcGIS.ADF.Web.DataSources. ArcGISServerc 程序集，并用 Add 按钮加入到 Selected Assemblies 中。加入完成后，选择 Finish 按钮关闭对话框。这时再编译程序则不会再有错误。

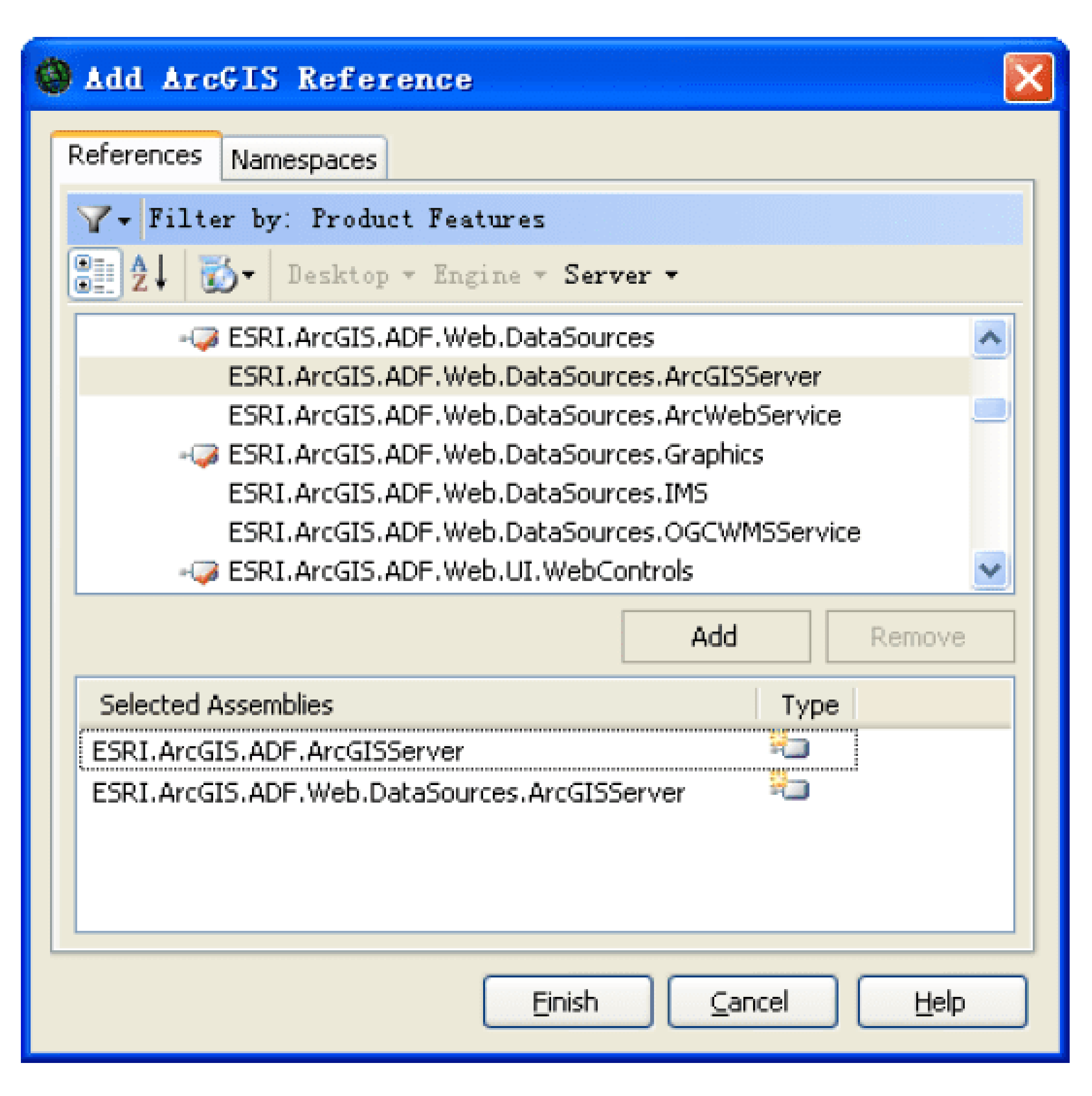


图 4.24　Add ArcGIS Reference 对话框

在清除了原高亮显示的要素后，要绘制新选择的要素。绘制要素则必须要有符号，为了 HighLightShow 方法看起来比较明了与简单，我们利用另一个静态方法创建一个符号，代码如下：

```
public static SimpleFillSymbol CreateSimpleFillSymbol()
{
    ESRI.ArcGIS.ADF.ArcGISServer.RgbColor rgb;
    rgb = new ESRI.ArcGIS.ADF.ArcGISServer.RgbColor();
```

```
        rgb.Red = 255;
        rgb.Green = 0;
        rgb.Blue = 0;
        rgb.AlphaValue = 255;
        SimpleLineSymbol lineSym = new SimpleLineSymbol();
        lineSym.Color = rgb;
        lineSym.Width = 1.0;
        lineSym.Style = esriSimpleLineStyle.esriSLSSolid;
        SimpleFillSymbol sfs;
        sfs = new SimpleFillSymbol();
        sfs.Style = esriSimpleFillStyle.esriSFSForwardDiagonal;
        sfs.Color = rgb;
        sfs.Outline = lineSym;

        return sfs;
    }
```

上述代码创建的是用红色斜线填充、红色边线的面符号。SimpleFillSymbol 类表示的是一个简单填充面符号。Style 属性指定的是其填充方式，我们使用的是 esriSFSForwardDiagonal，表示斜线填充，Color 属性表示填充的颜色，Outline 属性表示边线的线符号。

回到 HighLightShow 方法中，加入如下语句，调用上面的方法创建面符号：

```
SimpleFillSymbol sfs = CreateSimpleFillSymbol();
```

下面要实现的就是需要利用该符号绘制每个选中要素，代码如下：

```
foreach(DataTable dt in dtc)
{
    if (dt.Rows.Count == 0)
        continue;

    HighLightPolygon(mapDescription, dt, sfs);
}
```

在上述代码中，我们通过调用 HighLightPolygon 方法来实现真正的绘制工作。该方法的代码如下：

```
public static void HighLightPolygon(MapDescription mapDescription,
DataTable datatable, SimpleFillSymbol sfs)
{
    int hasCount = 0;
    if (mapDescription.CustomGraphics != null)
        hasCount = mapDescription.CustomGraphics.Length;
    ESRI.ArcGIS.ADF.ArcGISServer.GraphicElement[] ges;
    ges = new GraphicElement[hasCount + datatable.Rows.Count];
    CopyCustomGraphics(ges, mapDescription);

    int geoIndex = GeometryFieldIndex(datatable);
    for (int i = 0; i < datatable.Rows.Count; i++)
    {
        ESRI.ArcGIS.ADF.Web.Geometry.Polygon polygon =
datatable.Rows[i][geoIndex]
as ESRI.ArcGIS.ADF.Web.Geometry.Polygon;
        PolygonN ags_map_polyn;
```

```
        ags map polyn = ESRI.ArcGIS.ADF.Web.DataSources.ArcGISServer.
Converter.FromAdfPolygon(polygon);
        ESRI.ArcGIS.ADF.ArcGISServer.PolygonElement polyelement;
        polyelement = new ESRI.ArcGIS.ADF.ArcGISServer.PolygonElement();
        polyelement.Symbol = sfs;
        polyelement.Polygon = ags map polyn;
        ges[hasCount + i] = polyelement;
    }

    mapDescription.CustomGraphics = ges;
}
```

在上面代码的第一段中，由于我们选择的要素可能同时位于几个图层中，因此需要保留上次调用 HighLightPolygon 方法设置的 CustomGraphics 对象。然而由于该对象是一个数组，而不是一个 Array 对象，无法往该对象中加入新的要素，因此新增一个 CopyCustomGraphics 方法将该对象中的内容复制到临时变量 ges 中，该方法的代码如下：

```
public static void CopyCustomGraphics(GraphicElement[] ges,
MapDescription mapDescription)
{
    if (mapDescription.CustomGraphics != null)
    {
        for (int i = 0; i < mapDescription.CustomGraphics.Length; i++)
            ges[i] = mapDescription.CustomGraphics[i];
    }
}
```

在 HighLightPolygon 方法代码的第二段中，首先调用 GeometryFieldIndex 方法得到几何图形对象在数据表中字段的序号，其代码如下：

```
public static int GeometryFieldIndex(DataTable datatable)
{
    int geoIndex = -1;
    for (int i = 0; i < datatable.Columns.Count; i++)
    {
        if (datatable.Columns[i].DataType ==
 typeof(ESRI.ArcGIS.ADF.Web.Geometry.Geometry))
        {
            // 找到 Geometry 字段的序号
            geoIndex = i;
            break;
        }
    }

    return geoIndex;
}
```

然后通过一个循环，将数据表中的几何图形对象符号化为 Polyelement 对象，并加入到 ges 数组中。在该代码中，利用 datatable.Rows[i][geoIndex]得到几何图形对象，然后利用 Converter 类的 FromAdfPolygon 方法将其转换为 ESRI.ArcGIS.ADF.ArcGISServer.PolygonN 类型（关于 Converter 类的详细说明请参考 1.7 节）。最后将该 PolygonN 设置到 PolygonElement 对象的 Polygon 属性。

这里稍微解释一下为什么需要 Converter 类进行转换。这是由于我们使用了 MapDescription 值对象，因此也需要使用几何图形的值对象。

在 HighLightPolygon 方法代码的第三段中，将临时变量的值赋予 CustomGraphics，完成高亮显示。

再次回到 HighLightShow 方法中，加入如下语句，刷新地图：

```
RefreshMap(map, "NorthAmericaMap");
```

RefreshMap 方法的代码如下：

```
public static void RefreshMap(Map map, string resourceName)
{
    if (map.ImageBlendingMode == ImageBlendingMode.WebTier)
    {
        map.Refresh();
    }
    else if (map.ImageBlendingMode == ImageBlendingMode.Browser)
    {
        map.RefreshResource(resourceName);
    }
}
```

在该方法中首先判断地图对象的 ImageBlendingMode（地图图片融合方式）属性，如果该值为 WebTier，则刷新整个地图，否则只刷新指定的资源。

这里简要介绍一下地图图片的融合方成。

ArcGIS Server 提供了两种图形处理方式，一种称为浏览器端融合（Browser Blending），另一种成为 Web 层融合（Web-tier Blending）。

浏览器端融合的原理如图 4.25 所示。在该种方式下，来自不同 GIS 服务器的图片，不经过 Web 应用程序，直接发送给用户，在用户的浏览器端利用浏览器应用程序本身的功能融合为一张图片。

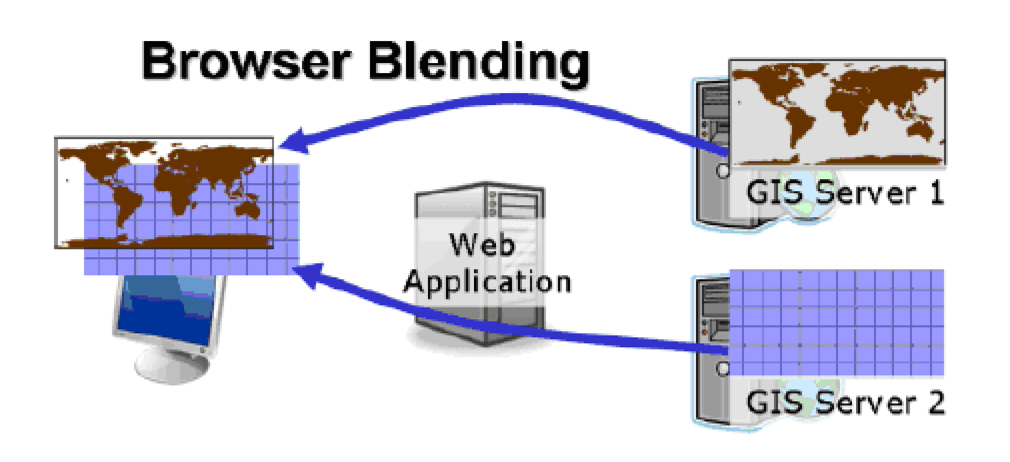


图 4.25　浏览器端融合过程

Web 层融合的原理如图 4.26 所示。在该种方式下，来自不同 GIS 服务器的图片在 Web 服务器上由 Web 应用程序进行融合，然后将结果发送到客户端的浏览器，进行显示。

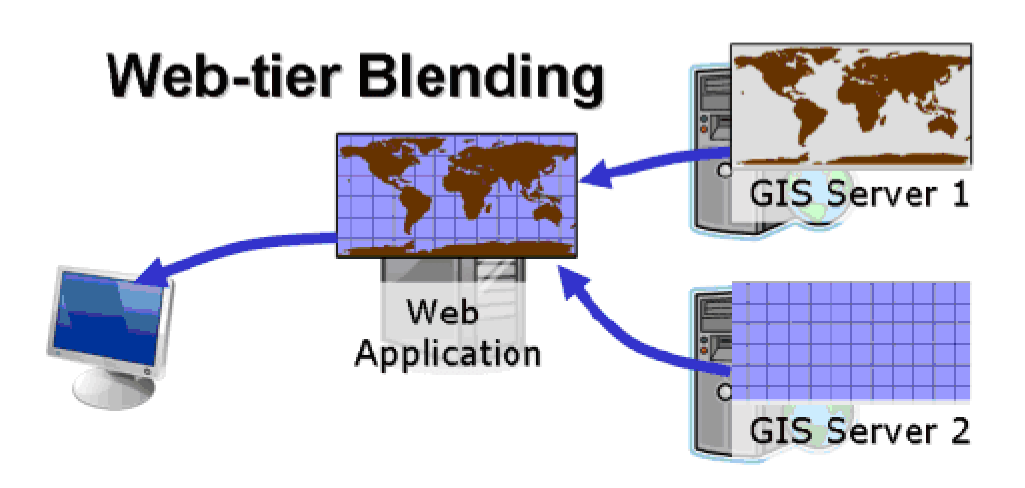


图 4.26　Web 层融合过程

完成 HighLightShow 方法后，切换到 IdentifyHelper 类的 ShowIdentifyResult 方法中，增加对 HighLightShow 方法的调用，ShowIdentifyResult 方法完整的代码如下：

```
public static void ShowIdentifyResult(Map map, DataTableCollection dtc)
{
    string returnstring = string.Empty;
    foreach (DataTable dt in dtc)
    {
        if (dt.Rows.Count == 0)
            continue;
        returnstring += GetHtmlFromDataTable(dt);
    }

    HighLightShow(map, dtc);

    returnstring = returnstring.Replace("\r\n", "");
    returnstring = returnstring.Replace("\n", "");
    string functionValue = "var theForm = document.forms[0];";
    functionValue += "theForm.FunctionValue.Value='" + returnstring + "';";
    functionValue += "open('IdentifyResult.htm', 'IdentifyResult');";
    AddJavaScriptCallback(map, functionValue);
}
```

运行程序，运行效果如图 4.27 所示。

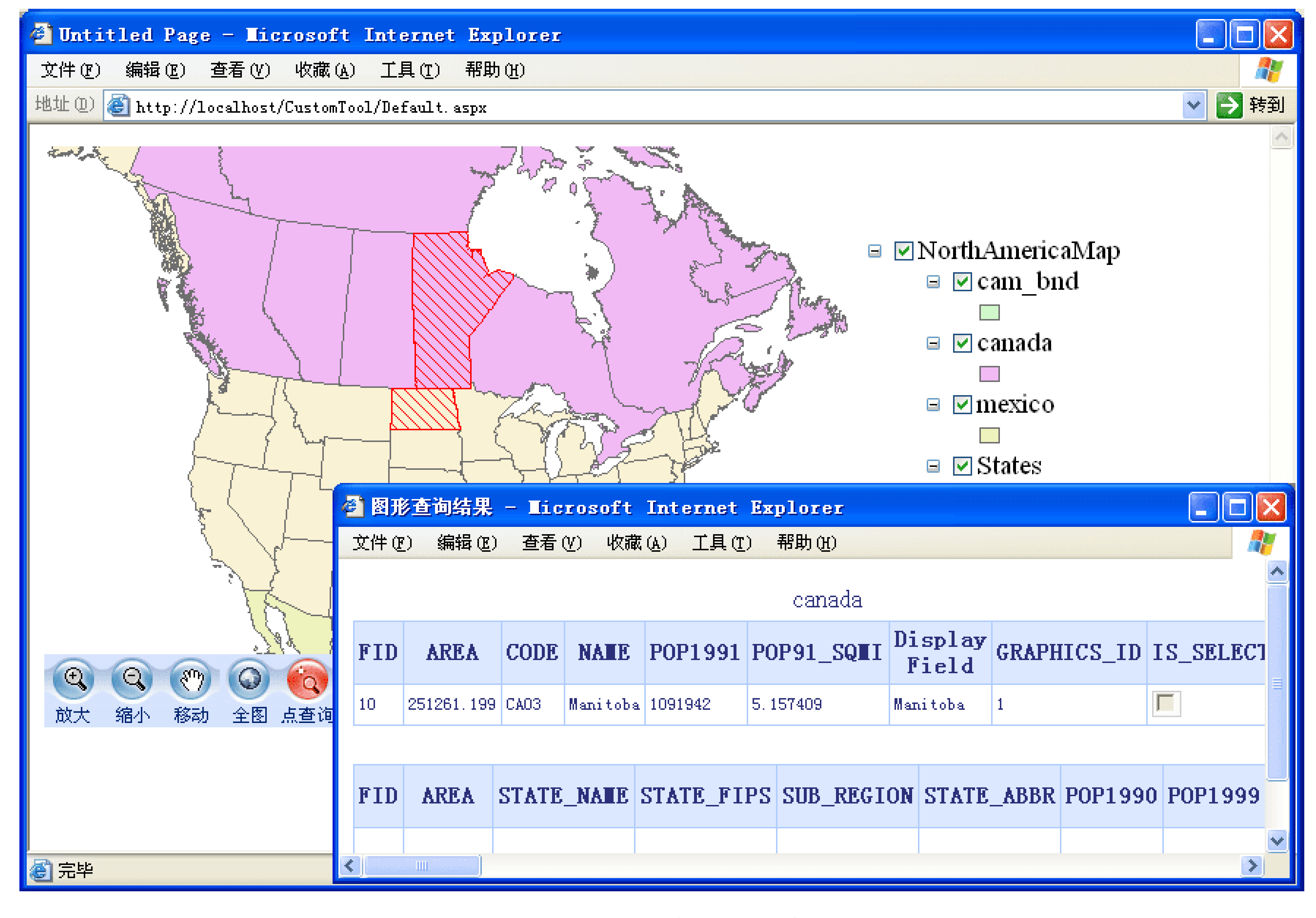


图 4.27 程序运行效果

4.4.4 自定义矩形查询工具

前面我们将点查询工具的实现代码分割成很小粒度的方法，为实现其他图形查询数据工具提供

了很好的基础，使得加入其他工具也变得非常简单。

要加入矩形查询工具，首先需要在 Default.aspx 页面的工具栏控件的<esri:ToolbarItems >与</esri:ToolbarItems >代码段之间再增加如下代码：

```
<esri:Tool ClientAction="DragRectangle" Name="RectIdentify"
   DefaultImage="~/Images/Toolbar/select_rect_1.jpg"
   HoverImage="~/Images/Toolbar/select rect 2.jpg"
   SelectedImage="~/Images/Toolbar/select rect 3.jpg"
   ServerActionAssembly="App Code"
   ServerActionClass="IdentifyRectangle"
   Text="RectIdentify"
   ToolTip="矩形查询" />
```

在上述代码中我们指定服务器端实现该矩形查询工具的类为 App_Code 文件夹下面的 `IdentifyRectangle` 类。

在 App_Code 文件夹下增加 `IdentifyRectangle` 类，该类的实现代码如下：

```
using ESRI.ArcGIS.ADF.Web.Geometry;
using ESRI.ArcGIS.ADF.Web.UI.WebControls;
using ESRI.ArcGIS.ADF.Web.UI.WebControls.Tools;

public class IdentifyRectangle : IMapServerToolAction {

   #region IServerAction Members

   void IMapServerToolAction.ServerAction(ToolEventArgs args) {
      Map mapCtrl = args.Control as Map;
      RectangleEventArgs rectargs = (RectangleEventArgs)args;
      System.Drawing.Rectangle myrect = rectargs.ScreenExtent;

      Point minpnt = Point.ToMapPoint(
         myrect.Left, myrect.Bottom, mapCtrl.Extent,
         (int)mapCtrl.Width.Value, (int)mapCtrl.Height.Value);
      Point maxpnt = Point.ToMapPoint(
         myrect.Right, myrect.Top, mapCtrl.Extent,
         (int)mapCtrl.Width.Value, (int)mapCtrl.Height.Value);
      Envelope mapPoly = new Envelope(minpnt, maxpnt);

      IdentifyHelper.Identify(mapCtrl, mapPoly);
   }

   #endregion
}
```

上述代码中指定 `IdentifyRectangle` 类实现 IMapServerToolAction 接口，这是工具类所必须的。

在 IMapServerToolAction.ServerAction 方法中通过参数 args 得到用户在地图上拖动的矩形的屏幕坐标范围，由于 Web ADF 没有提供矩形从屏幕坐标转换为地图坐标的方法，因此上述代码利用矩形的对角的两个点分别转换到地图坐标，然后利用 `Envelope` 类的构造函数构造一个地图坐标的矩形对象。最后调用 `IdentifyHelper` 类的 Identify 方法实现查询。

在 IdentifyHelper 类中加入如下代码：

```
public static void Identify(Map map,
ESRI.ArcGIS.ADF.Web.Geometry.Geometry mapGeometry)
{
}
```

我们将利用该方法实现矩形以及多边形、线查询，因此方法的第二个参数写成这些类的父类 Geometry，达到代码重用的目的。当然我们也可以利用 4.4.2 节中介绍的查询功能的 Identify 方法来实现，这里另写一个方法的目的只是为了向读者介绍不同的实现途径。

与点查询一样，在调用查询功能之前一定要判断资源是否具有查询功能，代码如下：

```
IGISFunctionality gisfunc = map.GetFunctionality("NorthAmericaMap");
if (gisfunc == null)
    return;

IGISResource gisresource = gisfunc.Resource;
bool supportquery =
gisresource.SupportsFunctionality(typeof(IQueryFunctionality));
if (!supportquery)
    return;
```

如果具有查询功能，则利用如下代码得到该查询功能：

```
IQueryFunctionality qfunc;
qfunc = gisresource.CreateFunctionality(typeof(IQueryFunctionality), null) as
IQueryFunctionality;
```

只能对可查询图层进行空间查询，因此在执行空间查询之前，需要得到可查询图层数组，代码如下：

```
string[] lIDs, lNames;
qfunc.GetQueryableLayers(null, out lIDs, out lNames);
```

接着依据传入的几何图形对象，构造一个空间查询对象，代码如下：

```
ESRI.ArcGIS.ADF.Web.SpatialFilter spatialfilter =
   new ESRI.ArcGIS.ADF.Web.SpatialFilter();
spatialfilter.ReturnADFGeometries = false;
spatialfilter.MaxRecords = 1000;
spatialfilter.Geometry = mapGeometry;
```

在上述代码中我们指定查询最多返回 1000 条记录，查询图形为传入的参数。此外还可以通过指定 SpatialFilter 类的 WhereClause 属性执行属性查询。

然后通过一个对可查询图层循环调用查询功能的 Query 方法，查询每个图层，代码如下：

```
System.Data.DataSet dataset = new System.Data.DataSet();
for (int i = 0; i < lIDs.Length; i++) {
    System.Data.DataTable datatable = qfunc.Query(
      null, lIDs[i], spatialfilter);
    if (datatable == null)
       continue;
    datatable.TableName = lNames[i];
    dataset.Tables.Add(datatable);
}
```

至此完成了图形查询要素工作，余下的是将查询到的要素的属性显示出来并高亮显示这些要

素，该功能已经在 4.4.2 与 4.4.3 两节中实现，直接调用即可。代码如下：

```
DataTableCollection dtc = dataset.Tables;
ShowIdentifyResult(map, dtc);
```

Identify 方法的完整代码如下：

```
public static void Identify(Map map,
ESRI.ArcGIS.ADF.Web.Geometry.Geometry mapGeometry)
{
    IGISFunctionality gisfunc = map.GetFunctionality("NorthAmericaMap");
    if (gisfunc == null)
        return;

    IGISResource gisresource = gisfunc.Resource;
    bool supportquery =
       gisresource.SupportsFunctionality(typeof(IQueryFunctionality));
    if (!supportquery)
        return;

    IQueryFunctionality qfunc;
    qfunc = gisresource.CreateFunctionality(
            typeof(IQueryFunctionality), null) as IQueryFunctionality;
    string[] lIDs, lNames;
    qfunc.GetQueryableLayers(null, out lIDs, out lNames);

    ESRI.ArcGIS.ADF.Web.SpatialFilter spatialfilter =
        new ESRI.ArcGIS.ADF.Web.SpatialFilter();
    spatialfilter.ReturnADFGeometries = false;
    spatialfilter.MaxRecords = 1000;
    spatialfilter.Geometry = mapGeometry;

    System.Data.DataSet dataset = new System.Data.DataSet();
    for (int i = 0; i < lIDs.Length; i++) {
        System.Data.DataTable datatable = qfunc.Query(
                                          null, lIDs[i], spatialfilter);
        if (datatable == null)
            continue;
        datatable.TableName = lNames[i];
        dataset.Tables.Add(datatable);
    }

    DataTableCollection dtc = dataset.Tables;
    ShowIdentifyResult(map, dtc);
}
```

编译并运行程序，查询效果如图 4.28 所示。

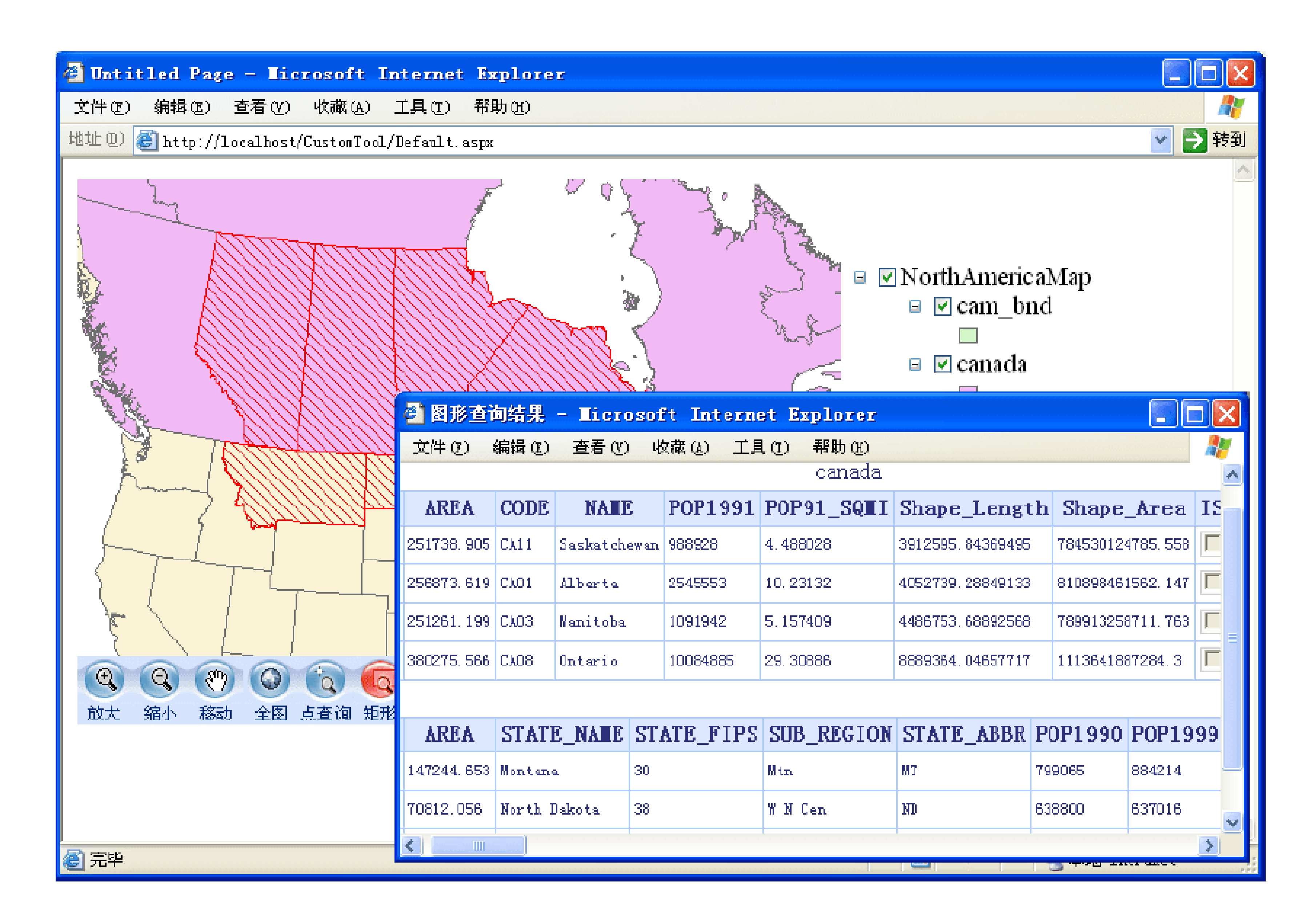


图 4.28 矩形查询效果

4.4.5 自定义多边形查询工具

在 Default.aspx 页面的工具栏控件的<esri:ToolbarItems >与</esri:ToolbarItems >代码段之间再增加如下代码，用于在工具栏上增加多边形查询工具按钮：

```
<esri:Tool ClientAction="Polygon" Name="PolygonIdentify"
    DefaultImage="~/Images/Toolbar/select polygon 1.jpg"
    HoverImage="~/Images/Toolbar/select polygon 2.jpg"
    SelectedImage="~/Images/Toolbar/select polygon 3.jpg"
    ServerActionAssembly="App_Code"
    ServerActionClass="IdentifyPolygon"
    Text="PolygonIdentify" ToolTip="多边形查询" />
```

在 App_Code 文件夹下增加 IdentifyPolygon 类，该类的实现代码如下：

```
using ESRI.ArcGIS.ADF.Web.Geometry;
using ESRI.ArcGIS.ADF.Web.UI.WebControls;
using ESRI.ArcGIS.ADF.Web.UI.WebControls.Tools;

public class IdentifyPolygon : IMapServerToolAction {
    #region IMapServerToolAction Members

    void IMapServerToolAction.ServerAction(ToolEventArgs args) {
        Map map = args.Control as Map;
        PolygonEventArgs polyArgs = (PolygonEventArgs)args;
        Polygon mapPoly = GeometryHelper.GetMapPolygon(map, polyArgs);
        IdentifyHelper.Identify(map, mapPoly);
    }
```

```
    #endregion
}
```

在上述代码中我们先通过 `GeometryHelper` 类的静态 GetMapPolygon 方法将屏幕坐标的多边形转换为地图坐标的多边形，然后直接调用在 4.4.4 节中实现的几何图形查询方法，即完成多边形查询。

在 App_Code 文件夹下增加 `GeometryHelper` 类，先在该类的头部加入如下命名空间的引用：

```
using ESRI.ArcGIS.ADF.Web.Geometry;
using ESRI.ArcGIS.ADF.Web.UI.WebControls;
```

然后加入如下代码，实现多边形坐标系统的转换：

```
public static ESRI.ArcGIS.ADF.Web.Geometry.Polygon GetMapPolygon(Map map,
PolygonEventArgs polyArgs)
{
    System.Drawing.Point[] screenpoly = polyArgs.Vectors;

    PointCollection pc = new ESRI.ArcGIS.ADF.Web.Geometry.PointCollection();
    foreach (System.Drawing.Point dpnt in screenpoly) {
        pc.Add(Point.ToMapPoint(dpnt, map.Extent,
            (int)map.Width.Value, (int)map.Height.Value));
    }

    Ring ring = new Ring();
    ring.Points = pc;
    RingCollection rings = new RingCollection();
    rings.Add(ring);
    Polygon mapPoly = new Polygon();
    mapPoly.Rings = rings;
    return mapPoly;
}
```

编译并运行程序，利用多边形查询工具查询地物要素，效果如图 4.29 所示。

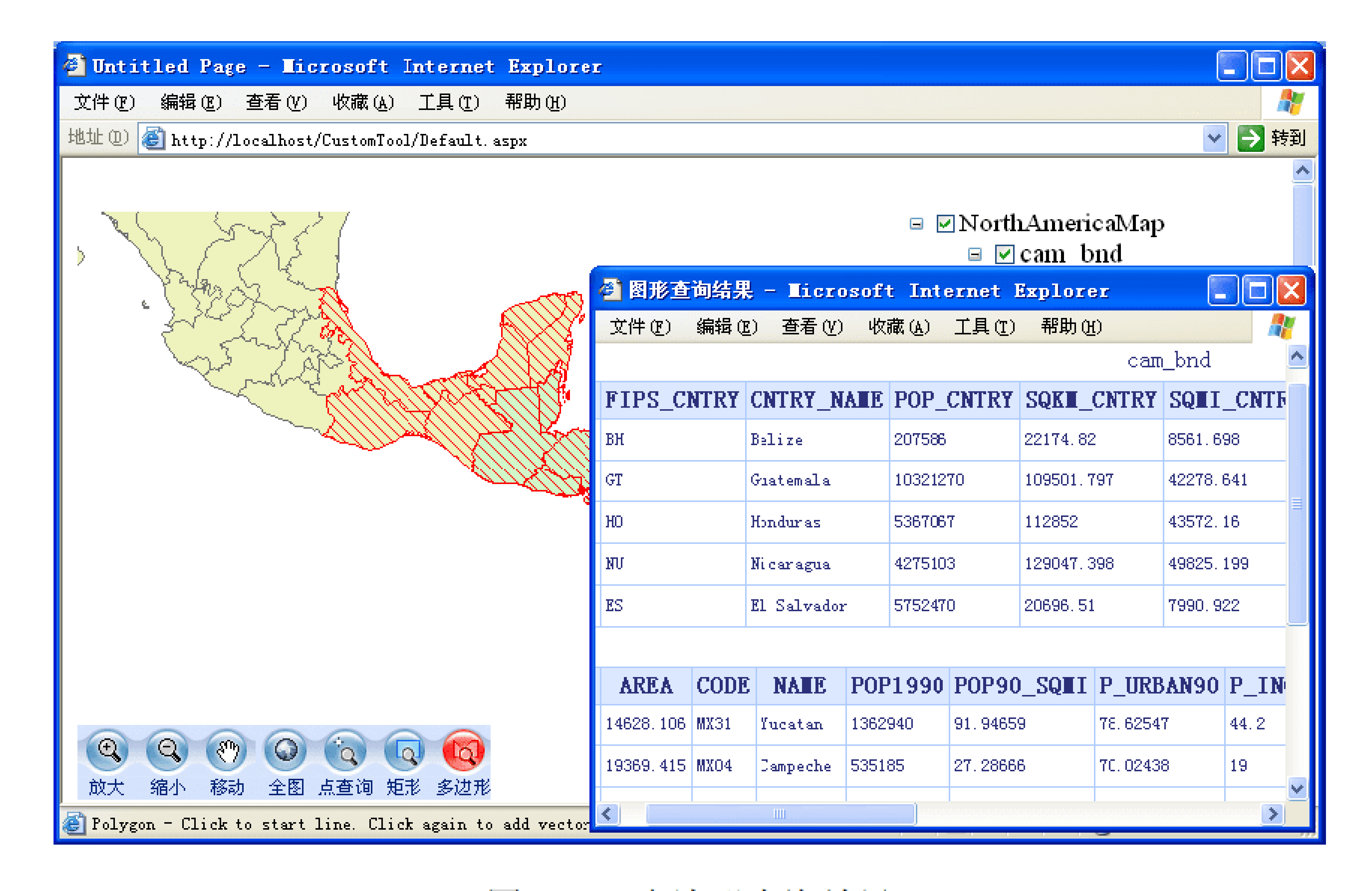


图 4.29　多边形查询效果

4.4.6 自定义圆查询工具

在 Default.aspx 页面的工具栏控件的<esri:ToolbarItems>与</esri:ToolbarItems>代码段之间再增加如下代码，用于在工具栏上增加圆查询工具按钮：

```
<esri:Tool ClientAction="Circle" Name="CircleIdentify"
    DefaultImage="~/Images/Toolbar/select circle 1.jpg"
    HoverImage="~/Images/Toolbar/select circle 2.jpg"
    SelectedImage="~/Images/Toolbar/select circle 3.jpg"
    ServerActionAssembly="App Code"
    ServerActionClass="IdentifyCircle"
    Text="CircleIdentify" ToolTip="圆查询" />
```

在 App_Code 文件夹下增加 `IdentifyRectangle` 类，该类的实现代码如下：

```
using ESRI.ArcGIS.ADF.Web.Geometry;
using ESRI.ArcGIS.ADF.Web.UI.WebControls;
using ESRI.ArcGIS.ADF.Web.UI.WebControls.Tools;

public class IdentifyCircle : IMapServerToolAction {
    #region IMapServerToolAction Members

    void IMapServerToolAction.ServerAction(ToolEventArgs args) {
        Map map = args.Control as Map;
        CircleEventArgs circleArgs = (CircleEventArgs)args;
        Polygon mapPoly = GeometryHelper.GetMapPolygon(map, circleArgs);
        IdentifyHelper.Identify(map, mapPoly);
    }

    #endregion
}
```

在 4.4.4 节中编写的 `IdentifyHelper` 类的 Identify 方法可实现几何图形的查询，我们只需直接调用即可。不过由于在 ESRI.ArcGIS.ADF.Web.Geometry.Geometry 类的子类中并没有一个表示圆的类，因此在调用 Identify 方法之前，先利用 `GeometryHelper` 类的 GetMapPolygon 方法，将一个多边形近似表示一个圆转换。

GeometryHelper 类的 GetMapPolygon 方法代码如下：

```
public static ESRI.ArcGIS.ADF.Web.Geometry.Polygon GetMapPolygon(
Map map, CircleEventArgs circleArgs) {
    System.Drawing.Point screenPointCenter = new System.Drawing.Point(
                (int)circleArgs.CenterPoint.X,
                (int)circleArgs.CenterPoint.Y);
    System.Drawing.Point screenPointBrink = new System.Drawing.Point(
                (int)circleArgs.CenterPoint.X,
                (int)circleArgs.CenterPoint.Y + (int)circleArgs.Radius);

    // 得到地图坐标中的半径
    Point mapPointCenter = Point.ToMapPoint(screenPointCenter, map.Extent,
        (int)map.Width.Value, (int)map.Height.Value);
    Point mapPointBrink = Point.ToMapPoint(screenPointBrink, map.Extent,
```

Web GIS

```
        (int)map.Width.Value, (int)map.Height.Value);
    double radius = Math.Sqrt(
        (mapPointCenter.X - mapPointBrink.X) *
        (mapPointCenter.X - mapPointBrink.X) +
        (mapPointCenter.Y- mapPointBrink.Y) *
        (mapPointCenter.Y - mapPointBrink.Y));

    GraphicsPath gpath = new GraphicsPath();
    gpath.AddEllipse((float)(mapPointCenter.X - radius),
        (float)(mapPointCenter.Y - radius),
        (float)(2 * radius), (float)(2 * radius));
    Matrix translateMatrix = new Matrix();
    translateMatrix.Translate(0, 0);

    float flattening = (float)(radius / 1000);
    gpath.Flatten(translateMatrix, flattening);

    PointCollection pc = new PointCollection();
    foreach (System.Drawing.PointF dpnt in gpath.PathPoints) {
        pc.Add(new ESRI.ArcGIS.ADF.Web.Geometry.Point(dpnt.X, dpnt.Y));
    }

    Ring ring = new Ring();
    ring.Points = pc;
    RingCollection rings = new RingCollection();
    rings.Add(ring);
    Polygon mapPoly = new Polygon();
    mapPoly.Rings = rings;
    return mapPoly;
}
```

编译并运行程序，选择圆查询工具，然后在地图上单击一点，然后移动鼠标，即可看到系统在地图上绘制圆，再次单击鼠标，选定圆的半径，完成圆查询。操作过程效果如图 4.30 所示。

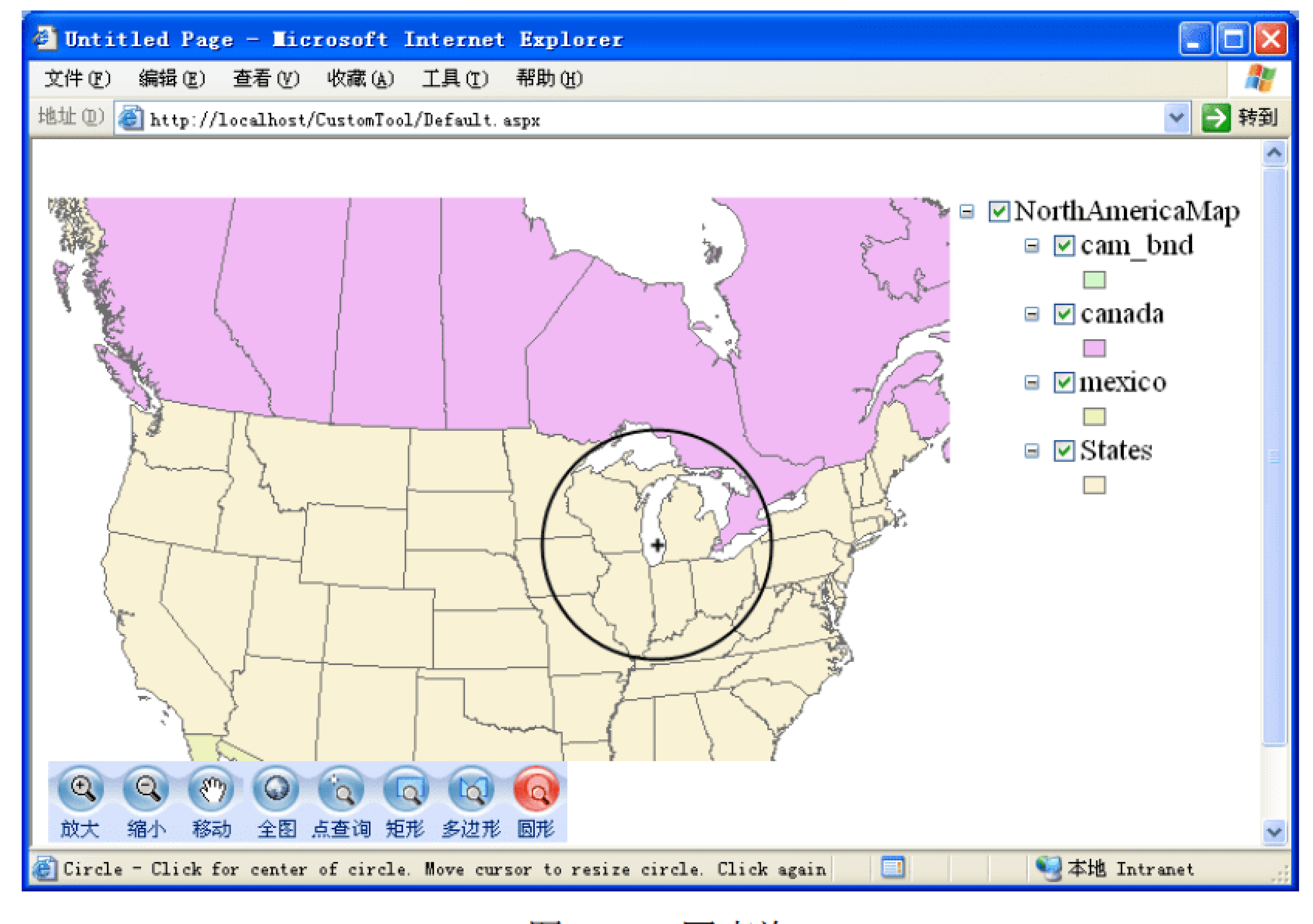


图 4.30　圆查询

Web GIS

4.4.7 自定义命令——清除高亮显示命令

前面几小节介绍了如何自定义工具，本小节介绍如何实现自定义命令。有时用户希望清除那些高亮显示的查询，通过一个命令就可实现。

同工具一样，首先需要在工具栏中增加一个按钮，只是该按钮的类型是命令。可直接在<esri:ToolbarItems>与</esri:ToolbarItems>代码段之间增加如下代码，在工具栏中增加名为ClearSelection的命令按钮：

```
<esri:Command ClientAction="" Name="ClearSelection"
   DefaultImage="~/Images/Toolbar/clearhighlight 1.jpg"
   HoverImage="~/Images/Toolbar/clearhighlight 2.jpg"
   SelectedImage="~/Images/Toolbar/clearhighlight 3.jpg"
   ServerActionAssembly="App Code"
   ServerActionClass="ClearSelection"
   Text="清除选择" ToolTip="清除选择" JavaScriptFile="" />
```

命令也需要在服务器端有一个类执行相关操作。与工具不同的是，该类需要实现的是IMapServerCommandAction接口，而不是IMapServerToolAction接口。

Web GIS

在App_Code文件夹下增加名为ClearSelection的类，该类的实现代码如下：

```
using ESRI.ArcGIS.ADF.Web.UI.WebControls;
using ESRI.ArcGIS.ADF.Web.UI.WebControls.Tools;
using ESRI.ArcGIS.ADF.Web.DataSources.ArcGISServer;

/// <summary>
/// 清除高亮度显示要素命令
/// </summary>
public class ClearSelection : IMapServerCommandAction
{
   #region IServerAction Members

   void IServerAction.ServerAction(ToolbarItemInfo info)
   {
      Map map = info.BuddyControls[0] as Map;
      ESRI.ArcGIS.ADF.Web.DataSources.ArcGISServer.MapFunctionality mf;
      mf = map.GetFunctionality("NorthAmericaMap") as MapFunctionality;
      MapDescription mapDescription = mf.MapDescription;
      if (mapDescription.CustomGraphics == null)
         return;
      if (mapDescription.CustomGraphics.Length == 0)
         return;
      mapDescription.CustomGraphics = null;
      IdentifyHelper.RefreshMap(map, "NorthAmericaMap");
   }

   #endregion
}
```

代码很简单，首先通过地图对象的 GetFunctionality 方法得到 NorthAmericaMap 资源的绘图功能，然后将绘图功能的 MapDescription 属性的 CustomGraphics 属性设置为 null，即可清除高亮显示要素，最后调用 IdentifyHelper 类的静态方法 RefreshMap 刷新地图，使客户端看到地图变化。

4.5　用属性查询图形

地理信息系统最常用的功能是通过图形查询属性以及通过属性查询图形，在 4.4 节中介绍的方法都属于通过几何图形来查询属性，本节将通过一个实例来介绍如何通过属性来查询图形。

本节实例要实现的是用户在文本框中输入条件，然后高亮显示满足该条件的图形。查询可以针对任何指定的图层，查询条件的字段可以是图层中的任一字段。

按照 4.1.3 节介绍的方法创建一个直接使用 Web 控件的 Web 应用程序，命名为 AttributeQuery，地图资源管理器、地图、Toc、工具栏控件的设置都同 4.1.3 小节的 FromControl。

4.5.1　资源内容查询

为了让用户能够选择需要查询的图层，需要提供一个下拉列表框，在该列表框中列出所有图层的名称，以便用户选择。因此首先要实现的是查询当前的地图资源中包含哪些图层。

首先需要在 Default.aspx 页面中的 Toc1 控件下方加入一标签控件（Lable，在工具箱的 Standard 选项卡中）Label1，将其 Text 属性设置为“查询图层：”。

注意，需要通过右键菜单的 Style 命令将 position 设置为 absolute，才能将控件拖到指定的位置。当然也可以通过直接修改页面代码实现。

然后在 Label1 控件的下方，加入一个 Div 控件（在工具箱的 HTML 选项卡中），将 ID 设置为 layerListDiv。我们将在该 Div 中用代码插入一个下拉列表框，显示所有图层名称。

为了实现部分页面刷新，本实例将充分利用 ASP.Net 2.0 的 Callback 机制。

要实现 Callback 机制，ASP.Net 的页面必须实现 ICallbackEventHandler，因此首先在 Default.aspx.cs 文件中，修改_Default 类的声明为如下代码：

```
public partial class _Default : System.Web.UI.Page, ICallbackEventHandler
```

由于我们希望程序运行后立即显示所有图层名称，所以需要页面加载时执行查询。因此在 Default.aspx 页面代码的最后加入如下代码：

```
<script type="text/javascript" language="javascript">
    function window.onload()
    {
        GetResourceContent();
    }
</script>
```

该 JavaScript 代码表示当页面加载时执行 GetResourceContent 函数。在页面的<head>与</head>

之间加入如下代码，实现 GetResourceContent 函数：

```
<script  language="javascript" type="text/javascript">
    function GetResourceContent()
    {
        var message = 'ActiveType=GetResourceContent';
        var context = 'Page1';
        <%=getResourceContentCallBack%>
    }
</script>
```

在上述代码中，我们指定 ActiveType（标识当前函数）为 GetResourceContent，然后执行实际的回调脚本，该脚本是利用 GetCallbackEventReference 方法生成的。切换到 Default.aspx.cs 文件中，声明如下变量：

```
public string getResourceContentCallBack;
```

并在 Page_Load 方法中增加如下代码，生成调用的客户端脚本：

```
getResourceContentCallBack = Page.ClientScript.GetCallbackEventReference(
                              this, "message", "processCallbackResult",
                              "context", "postBackError", true);
```

上述代码表明，传入的参数用 message 变量表示，上下文变量用 context 表示，如果成功执行服务器端代码，返回到客户端后执行 processCallbackResult 函数，否则执行 postBackError 函数。这两个函数都是 Web ADF 自身带的。

下面要实现的就是 ICallbackEventHandler 接口要求的两个方法。定位到"ICallbackEventHandler Members"代码区域，在其中首先加入如下一用于保存传入参数的字符串变量：

```
private string callbackArg;
```

服务器首先执行的是 RaiseCallbackEvent 方法，在该方法中，将传入参数保存到类的字段 callbackArg 中，代码如下：

```
void ICallbackEventHandler.RaiseCallbackEvent(string eventArgument) {
    callbackArg = eventArgument;
}
```

服务器端接着执行的是 GetCallbackResult，真正实现代码在该方法中。代码如下：

```
string ICallbackEventHandler.GetCallbackResult() {

    // 将传入参数依据&分割符分到 querystring 变量中
    Array keyValuePairs = callbackArg.Split("&".ToCharArray());
    NameValueCollection queryString = new NameValueCollection();
    string[] keyValue;
    string response = "";
    if (keyValuePairs.Length > 0) {
        for (int i = 0; i < keyValuePairs.Length; i++) {
            keyValue =
               keyValuePairs.GetValue(i).ToString().Split("=".ToCharArray());
            queryString.Add(keyValue[0], keyValue[1]);
```

```
            }
        }
        else {
            keyValue = callbackArg.Split("=".ToCharArray());
            if (keyValue.Length > 0)
                queryString.Add(keyValue[0], keyValue[1]);
        }

        // 针对参数中指定的 ActiveType 不同执行不同操作
        string controlType = queryString["ActiveType"];
        string eventArg = queryString["EventArg"];
        switch (controlType) {
            // 获取资源图层信息
            case "GetResourceContent":
                response = GisFunctionality.GetResourceContent(Map1);
                break;
            default:
                break;
        }

        return response;
    }
```

在上述代码中，我们首先依据“&”分割符，将传到服务器端的参数转换为 NameValueCollection 类型（可通过键或索引访问的关联 String 键和 String 值的集合），然后依据客户端指定的不同类型，即 ActiveType 的值不同，执行不同的操作。

对于执行资源图层信息查询，我们调用的是 GisFunctionality 类的 GetResourceContent 静态方法。

在 App_Code 路径中加入名为 GisFunctionality 的类。在该类的头部加入如下命名空间引用代码：

```
    using ESRI.ArcGIS.ADF.Web.UI.WebControls;
    using ESRI.ArcGIS.ADF.Web.DataSources.ArcGISServer;
```

注意，仍然需要读者按照 4.4.3 节中介绍的方法加入 ArcGISServer 的引用。

GetResourceContent 方法的实现代码如下：

```
    public static string GetResourceContent(Map map) {
        ESRI.ArcGIS.Carto.IMapLayerInfos mapLayers = GetMapLayerInfos(map);

        System.Text.StringBuilder strBld = new System.Text.StringBuilder();
        strBld.Append("<select id='LayerList' onchange='ShowFieldInfo()'>");

        for (int i = 0; i < mapLayers.Count; i++) {
            ESRI.ArcGIS.Carto.IMapLayerInfo linfo = mapLayers.get Element(i)
                as ESRI.ArcGIS.Carto.IMapLayerInfo;
            strBld.Append("<option value=\"" + linfo.ID + "\">");
            strBld.Append(linfo.Name + "</option>");
        }
        strBld.Append("</select>");
```

```
        CallbackResult cr = new CallbackResult("div", "layerListDiv",
            "innercontent", strBld.ToString());
        map.CallbackResults.Add(cr);
        return map.CallbackResults.ToString();
    }
```

在该方法用首先调用 GetMapLayerInfos 方法，得到地图图层信息对象，然后依据地图图层信息构造一个字符串表示的下拉列表框，接着把该字符串通过 CallbackResult 类的构造函数构造为 ID 为 layerListDiv 的 div 控件，并将该 CallbackResult 加入到地图的 CallbackResults 属性中，最后将该属性返回。我们在 4.4.2 节中已经详细介绍了 CallbackResult 的原理，还不了解的读者可再回头看看。

GetMapLayerInfos 方法的代码如下：

```
    private static ESRI.ArcGIS.Carto.IMapLayerInfos GetMapLayerInfos(Map map)
    {
        MapFunctionality mf =
           (MapFunctionality)map.GetFunctionality("NorthAmericaMap");
        MapResourceLocal mr;
        mr = mf.MapResource as MapResourceLocal;
        ESRI.ArcGIS.Server.IServerContext context =
           mr.ServerContextInfo.ServerContext;
        ESRI.ArcGIS.Carto.IMapServer server =
           context.ServerObject as ESRI.ArcGIS.Carto.IMapServer;
        ESRI.ArcGIS.Carto.IMapServerInfo mapInfo =
            server.GetServerInfo(server.DefaultMapName);
        return (ESRI.ArcGIS.Carto.IMapLayerInfos)mapInfo.MapLayerInfos;
    }
```


在上述的代码中，首先利用地图对象的 GetFunctionality 方法得到 NorthAmericaMap 资源的绘图功能，然后由此功能得到类型为 MapResourceLocal 的本地地图资源，然后由该资源得到地图服务的上下文对象，最后得到地图图层信息对象。

编译并运行程序，结果如图 4.31 所示。

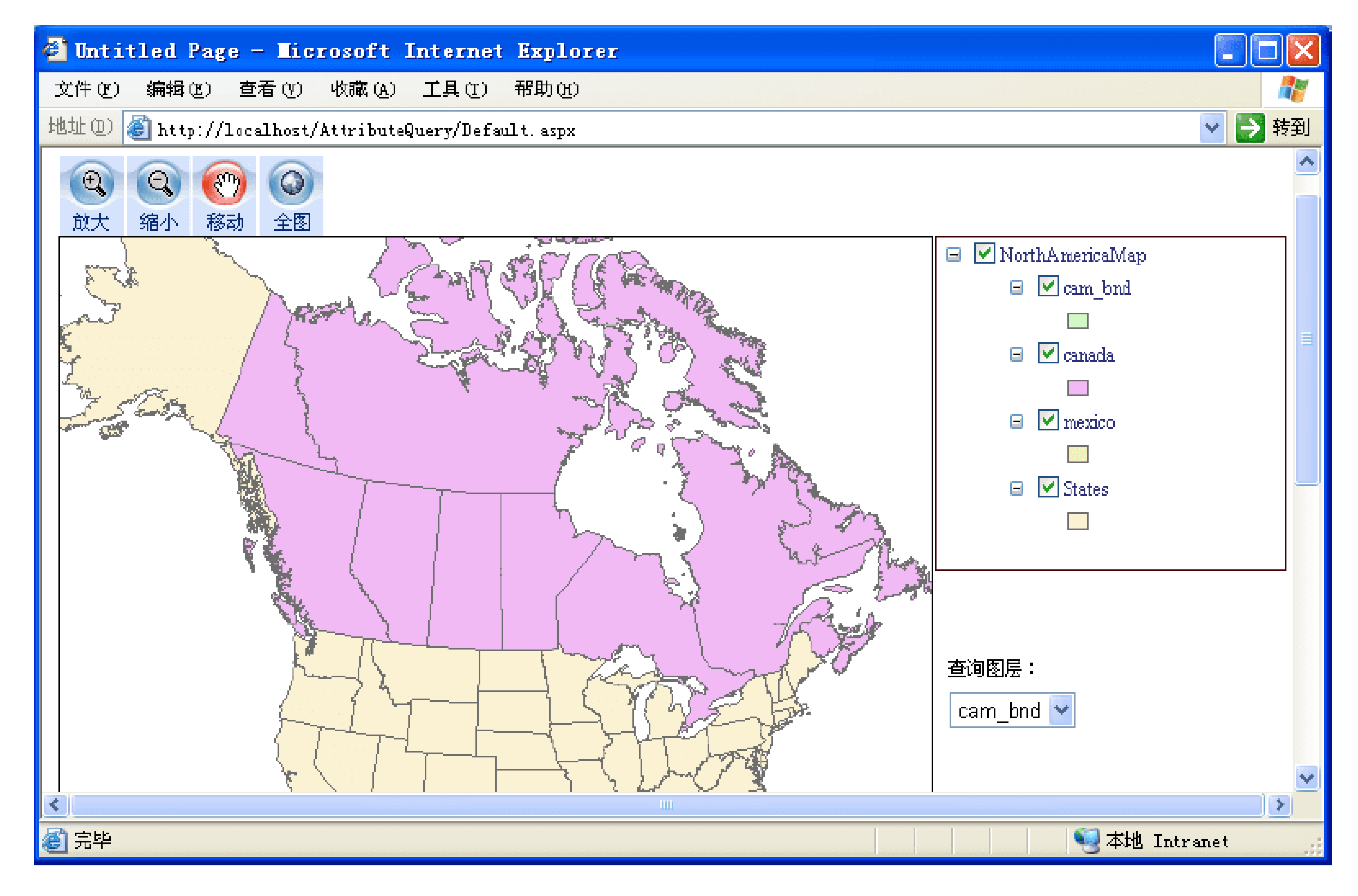


图 4.31　查询地图资源的图层信息

4.5.2　图层字段信息查询

要通过属性查询图形，就必须事先知道图层中包含哪些字段。

这一节要实现的就是当用户通过下列列表框选择某图层时，将图层中的所有字段及其数据类型以一个表格的方式表现出来。首先需要在 Default.aspx 页面中的 layerListDiv 控件下方加入一名为 fieldListDiv 的 Div 控件。然后切换到页面<head>与</head>之间的代码，在 GetResourceContent 函数下面加入如下函数：

```
function ShowFieldInfo()
{
    var o = document.getElementById("LayerList");
    var message = 'ActiveType=ShowFieldInfo&EventArg=';
    message += o.value;
    var context = 'Page1';
    <%=showFieldInfoCallBack%>
}
```

在 GetResourceContent 方法中构造下拉列表框控件时，指定当该控件发生 onchange 事件是调用 ShowFieldInfo()函数。

在上述代码中，我们指定 ActiveType 为 ShowFieldInfo，并设置 EventArg（即事件参数）为当前用户选择的图层的 ID 值。

切换到 Default.aspx.cs 文件中，在_Default 类中再加入如下公用字段：

```
public string showFieldInfoCallBack;
```

然后在 Page_Load 方法中增加如下代码，生成调用的客户端脚本：

```
showFieldInfoCallBack = Page.ClientScript.GetCallbackEventReference(this,
    "message", "processCallbackResult", "context", "postBackError", true);
```

然后在 ICallbackEventHandler.GetCallbackResult()方法的针对不同 ActiveType 执行不同操作的 switch 部分代码中加入如下语句，执行图层字段查询：

```
case "ShowFieldInfo":
    response = GisFunctionality.ShowFieldInfo(Map1, eventArg);
    break;
```

上面的代码表明，真正实现查询的是 GisFunctionality 类的 ShowFieldInfo 静态方法。

切换到 GisFunctionality.cs 文件中，加入 ShowFieldInfo 方法，其代码如下：

```
public static string ShowFieldInfo(Map map, string layerIndex){

    ESRI.ArcGIS.Carto.IMapLayerInfos mapLayers = GetMapLayerInfos(map);

    DataColumn fieldNameColumn = new DataColumn("字段名");
    fieldNameColumn.DataType = System.Type.GetType("System.String");
    DataColumn fieldDataTypeColumn = new DataColumn("字段类型");
    fieldDataTypeColumn.DataType = System.Type.GetType("System.String");
```

```
        System.Data.DataTable table = new DataTable();
        table.Columns.Add(fieldNameColumn);
        table.Columns.Add(fieldDataTypeColumn);

        for (int i = 0; i < mapLayers.Count; i++)
        {
            ESRI.ArcGIS.Carto.IMapLayerInfo linfo = mapLayers.get Element(i);
            if (!linfo.ID.ToString().Equals(layerIndex))
                continue;

            ESRI.ArcGIS.Geodatabase.IFields fields = linfo.Fields;
            for (int j = 0; j < fields.FieldCount; j++)
            {
                ESRI.ArcGIS.Geodatabase.IField field = fields.get Field(j);
                string[] rowValue = new string[2];
                rowValue[0] = field.Name;
                rowValue[1] = field.Type.ToString();
                table.Rows.Add(rowValue);
            }
        }

        string returnstring = GetHtmlFromDataTable(table);
        CallbackResult cr = new CallbackResult("div", "fieldListDiv",
                "innercontent", returnstring);
        map.CallbackResults.Add(cr);
        return map.CallbackResults.ToString();
    }
```

在上面的代码中，首先调用 GetMapLayerInfos 方法得到图层信息。然后构造了一个数据表对象，该数据表中包含两个字段，一个用于保存图层字段的名称，另一个保存图层字段的数据类型。然后通过循环将用户指定图层的字段信息填充到数据表中。接着调用 GetHtmlFromDataTable 方法依据数据表构造一个表格形式的 HTML 字符串。最后通过 CallbackResult 类将该字符串表示表格填充到名为 fieldListDiv 的 Div 控件中。

GetHtmlFromDataTable 方法的代码如下：

```
    public static string GetHtmlFromDataTable(DataTable dt) {
        GridView gd = new GridView();
        gd.ToolTip = dt.TableName;
        gd.Caption = dt.TableName;
        gd.DataSource = dt;
        gd.DataBind();
        gd.Visible = true;
        gd.BorderWidth = 0;
        gd.CssClass = "list-line";
        gd.CellPadding = 3;
        gd.CellSpacing = 1;
        gd.HeaderStyle.CssClass = "barbg";
        gd.HeaderStyle.HorizontalAlign = HorizontalAlign.Center;
        gd.RowStyle.CssClass = "listbg";
```

```
        string returnString = string.Empty;
        using (System.IO.StringWriter sw = new System.IO.StringWriter()) {
            HtmlTextWriter htw = new HtmlTextWriter(sw);
            gd.RenderControl(htw);
            htw.Flush();
            string tempStr = sw.ToString();
            returnString += tempStr;
        }
        return returnString;
    }
```

该方法利用 GridView 与 HtmlTextWriter 类将数据表中的内容转换为表格格式的 HTML 字符串。

为了使表格美观，需要利用本书提供的样式文件，在 Default.aspx 页面的 title 标签下面加入如下代码：

```
    <link href="css/style1.css" type="text/css" rel="stylesheet"/>
```

编译并运行程序，这时在下拉列表框中选择不同的图层，可看到该图层的所有字段及其数据类型的列表，效果如图 4.32 所示。

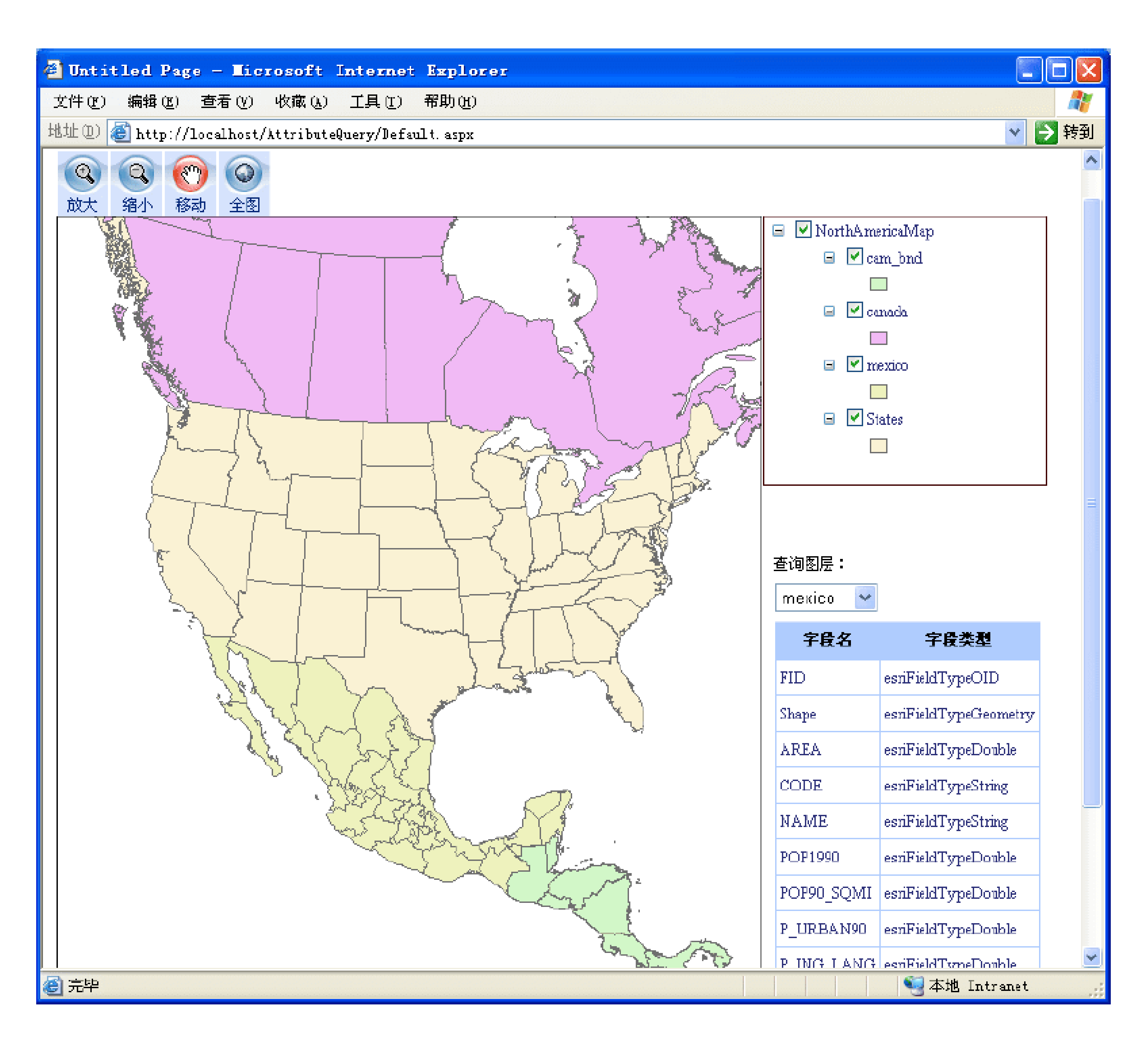


图 4.32　图层字段信息查询

通过运行该应用程序，读者可以看到在利用了 Callback 机制以后，没有了原来那种剧烈的页面的刷新，用户体验得到了非常大的提高。

此外，对于 Web 地理信息系统，另一个需要特别强调的是系统的响应速度。有许多方法来提

升系统的速度，例如利用缓存。

对于上面这个应用程序，我们可以将查询得到的图层字段信息缓存起来，这样第一个用户调用时，我们需要通过 ShowFieldInfo 方法从地图服务器上获取信息，当第二个用户或第二次调用时，我们可以直接返回缓存的信息，节省了从地图服务器中查询的时间，从而提高了响应效率。

ASP.NET 提供了一个强大的、便于使用的缓存机制，用于将需要大量服务器资源来创建的对象存储在内存中。缓存这些类型的资源会大大改进应用程序的性能。缓存是由 Cache 类实现的；缓存实例是每个应用程序专用的。缓存生存期依赖于应用程序的生存期；重新启动应用程序后，将重新创建 Cache 对象。

设计 Cache 类是为了便于使用。可以将项放置在 Cache 中，并在以后使用简单的键/值对来检索这些项。Cache 类提供了强大的功能，允许自定义如何缓存项以及将它们缓存多长时间。例如，当缺乏系统内存时，缓存会自动移除很少使用的或优先级较低的项以释放内存。该技术也称为清理，这是缓存确保过期数据不使用宝贵的服务器资源的方式之一。当执行清理时，可以指示 Cache 给予某些项比其他项更高的优先级。若要指示项的重要性，可以在使用 Add 或 Insert 方法添加项时指定一个 CacheItemPriority 枚举值。当使用 Add 或 Insert 方法将项添加到缓存时，您还可以建立项的过期策略。可以通过使用 DateTime 值指定项的确切过期时间（绝对过期时间），来定义项的生存期。也可以使用 TimeSpan 值指定一个弹性过期时间，弹性过期时间允许根据项的上次访问时间来指定该项过期之前的运行时间。一旦项过期，便将它从缓存中移除。试图检索它的值的行为将返回 null，除非该项被重新添加到缓存中。

此外，ASP.NET 允许根据外部文件、目录（文件依赖项）或另一个缓存项（键依赖项）来定义缓存项的有效性。如果具有关联依赖项的项发生更改，缓存项便会失效并从缓存中移除。可以使用该技术在项的数据源更改时从缓存中移除这些项。

下面的代码演示了如何利用 ASP.NET 提供的缓存机制。首先在 GetCallbackResult 方法的最前面加入如下代码，判断是否存在某一缓存，如果存在（第二次调用相同操作），则直接将缓存结果返回，如果不存在（首次调用），则不影响程序执行：

```
// 判断是否有相应的缓存信息
string cachedResponse = Cache[callbackArg] as string;
if (cachedResponse != null)
{
    return cachedResponse;
}
```

然后在 GetCallbackResult 方法返回 response 字符串代码之前，加入如下代码，用于将查询结果保存到缓存中：

```
TimeSpan cacheDuration = new TimeSpan(0, 0, 999999);
Cache.Add(callbackArg, response, null,
    DateTime.Now.Add(cacheDuration),
    System.Web.Caching.Cache.NoSlidingExpiration,
    System.Web.Caching.CacheItemPriority.NotRemovable, null);
```

上述代码通过 Cache 的 Add 方法将查询结果缓存起来。该方法的第一个参数是用于引用该项的缓存键；第二个参数是要添加到缓存的项；第三个参数表示文件依赖项或缓存键依赖项，当任何依赖项更改时，该对象即无效，并从缓存中移除。如果没有依赖项，则此参数为空引用（在 Visual

Basic 中为 Nothing）；第四个参数是所添加对象将过期并被从缓存中移除的时间。如果使用可调过期，则该参数必须为 NoAbsoluteExpiration；第五个参数表示最后一次访问所添加对象时与该对象过期时之间的时间间隔。如果该值等效于 20 分钟，则对象在最后一次被访问 20 分钟之后将过期并从缓存中移除。如果使用绝对过期，则该参数必须为 NoSlidingExpiration；第六个参数是对象的相对优先级，由 CacheItemPriority 枚举表示。缓存在退出对象时使用该值；具有较低优先级的对象在具有较高优先级的对象之前被从缓存移除；最后一个参数表示在从缓存中移除对象时所调用的委托（如果提供）。当从缓存中删除应用程序的对象时，可使用它来通知应用程序。

GetCallbackResult 方法的完整代码如下：

```
    string ICallbackEventHandler.GetCallbackResult()
    {
        // 判断是否有相应的缓存信息
        string cachedResponse = Cache[callbackArg] as string;
        if (cachedResponse != null)
        {
            return cachedResponse;
        }

        // 将传入参数依据&分割符分到 querystring 变量中
        Array keyValuePairs = callbackArg.Split("&".ToCharArray());
        NameValueCollection queryString = new NameValueCollection();
        string[] keyValue;
        string response = "";
        if (keyValuePairs.Length > 0)
        {
            for (int i = 0; i < keyValuePairs.Length; i++)
            {
                keyValue                                                =
keyValuePairs.GetValue(i).ToString().Split("=".ToCharArray());
                queryString.Add(keyValue[0], keyValue[1]);
            }
        }
        else
        {
            keyValue = callbackArg.Split("=".ToCharArray());
            if (keyValue.Length > 0)
                queryString.Add(keyValue[0], keyValue[1]);
        }

        // 针对参数中指定的 ActiveType 不同执行不同操作
        string controlType = queryString["ActiveType"];
        string eventArg = queryString["EventArg"];
        switch (controlType)
        {
            case "GetResourceContent":
                response = GisFunctionality.GetResourceContent(Map1);
                break;
            case "ShowFieldInfo":
```

```
            response = GisFunctionality.ShowFieldInfo(Map1, eventArg);
            break;
        default:
            break;
    }

    TimeSpan cacheDuration = new TimeSpan(0, 0, 999999);
    Cache.Add(callbackArg, response, null,
        DateTime.Now.Add(cacheDuration),
        System.Web.Caching.Cache.NoSlidingExpiration,
        System.Web.Caching.CacheItemPriority.NotRemovable, null);

    return response;
}
```

编译并运行程序。可以通过设置断点来确定缓存机制是否起到了作用。

正由于缓存可以很大提高系统响应效率，因此有不少程序员在这方面做了不少工作。其中 Steven Smith 开发了一个缓存管理工具，读者可以从 http://aspalliance.com/cachemanager/地址下载，也可以从本书的网上随附源代码文件中 AttributeQuery 目录的 bin 文件夹中找到（AspAlliance.CacheManager.dll）。

利用工程的右键菜单的 Add ASP.NET Folder 菜单的 Bin 命令，在工程中加入一名为 Bin 的文件夹，将 AspAlliance.CacheManager.dll 文件拷贝到该文件夹中。

在 Web.Config 文件的<system.web>与</system.web>之间加入如下代码，声明访问路径为 CacheManager.axd 的地址，使用 CacheManager 类来处理：

```
<httpHandlers>
  <add verb="*"
       path="CacheManager.axd"
       type="AspAlliance.CacheManager.CacheManagerPageFactory,
AspAlliance.CacheManager" />
</httpHandlers>
```

该工具封装得如此的好，以至通过上述两个步骤就可以使用该工具来管理应用程序的缓存了。

编译并运行程序，从下拉列表框中选择一个图层，查看该图层的字段信息。从前面的代码可以知道，上述操作缓存了资源内容信息（程序启动后自动执行）与选择图层的字段信息。在地址栏中输入 http://localhost/AttributeQuery/CacheManager.axd 地址，便可看到缓存管理工具页面，切换到 View Cache Entries 页面，便可我们上面缓存信息的列表，如图 4.33 所示（列表中的第一行是 ASP.NET 为调试而进行的缓存信息）。

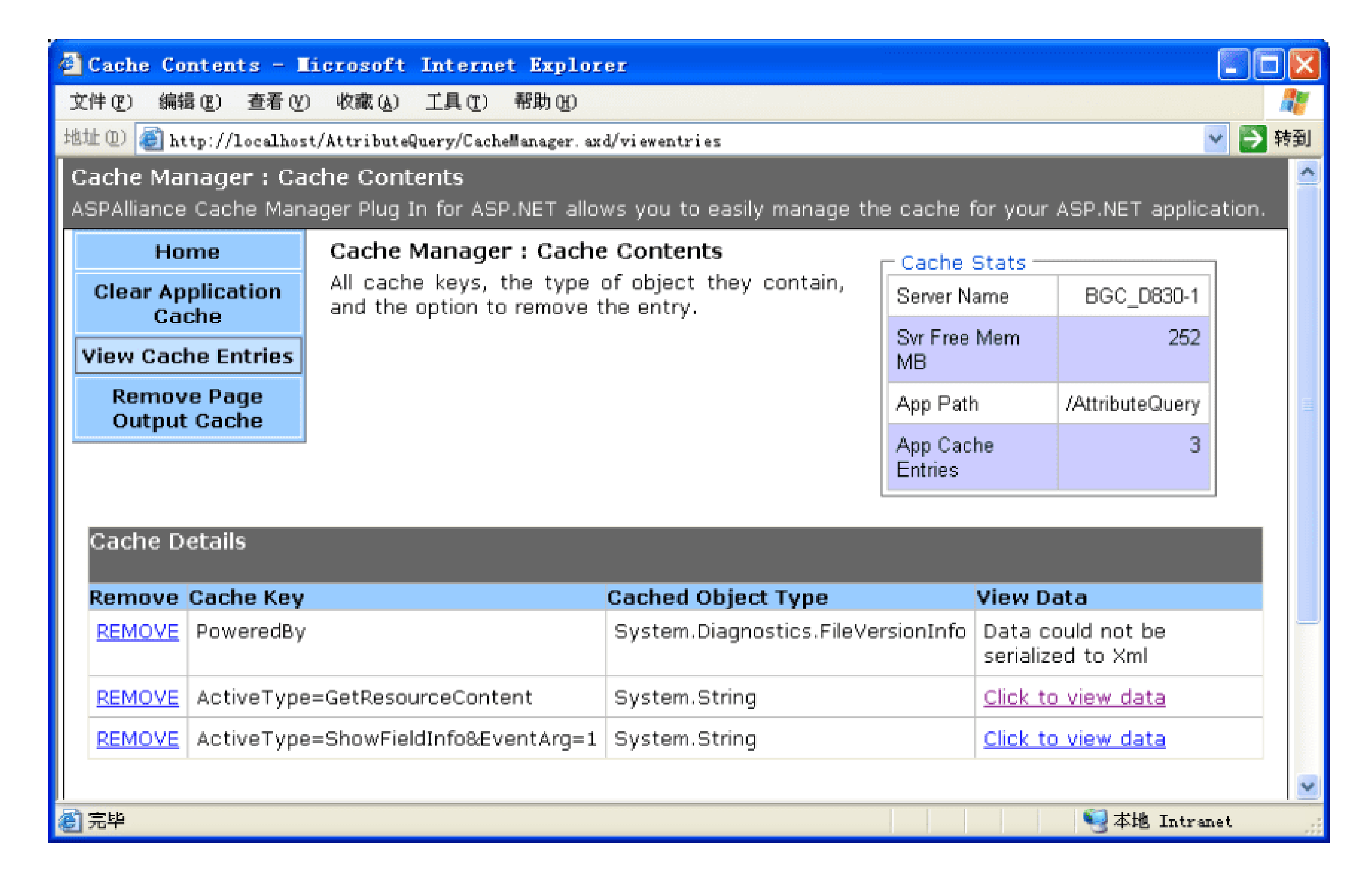


图 4.33　缓存管理工具页面

通过列表的 REMOVE 链接可以清除某一个缓存，还可以通过 click to view data 链接查看缓存的数据。

4.5.3　属性查询

知道每个图层中包含的字段及其数据类型后，便可实现属性查询了。

首先在 Default.aspx 页面中 Toc1 控件的下方与 Label1 控件的上方，加入一个标签控件 Label2，将其 Text 属性设置为“查询条件：”。

在 Label2 控件的下方与 Label1 控件的上方，从工具箱中的 HTML 选项卡中加入一文本框控件，将其 id 设置为“queryCondition”。

在上述文本框控件的下方与 Label1 控件的上方，从工具箱中的 HTML 选项卡中加入一个按钮控件，将其 id 设置为“Query”，value 设置为“查询”，onclick 设置为“AttributeQuery()”。

由于查询得到的要素需要高亮显示（这里介绍不同于 4.4.3 节的实现方法），我们准备将其绘制到另一个独立的图形资源的图层中，因此需要通过地图资源管理器的 ResourceItems 属性设置对话框加入另一资源。

在图 4.34 所示的 MapResourceItem Collection Editor 对话框中，先通过 Add 按钮，加入一资源，并将其 Name 属性设置为“SelectedFeature”，通过向上的箭头按钮，将该资源的位置调整到最上面。然后单击其 Definition 属性，将显示一个椭圆按钮，单击该按钮，弹出 Map Resource Definition Editor 对话框，在 Type 的下拉列表框中选择“GraphicsLayer”。选择 OK 按钮保存设置。

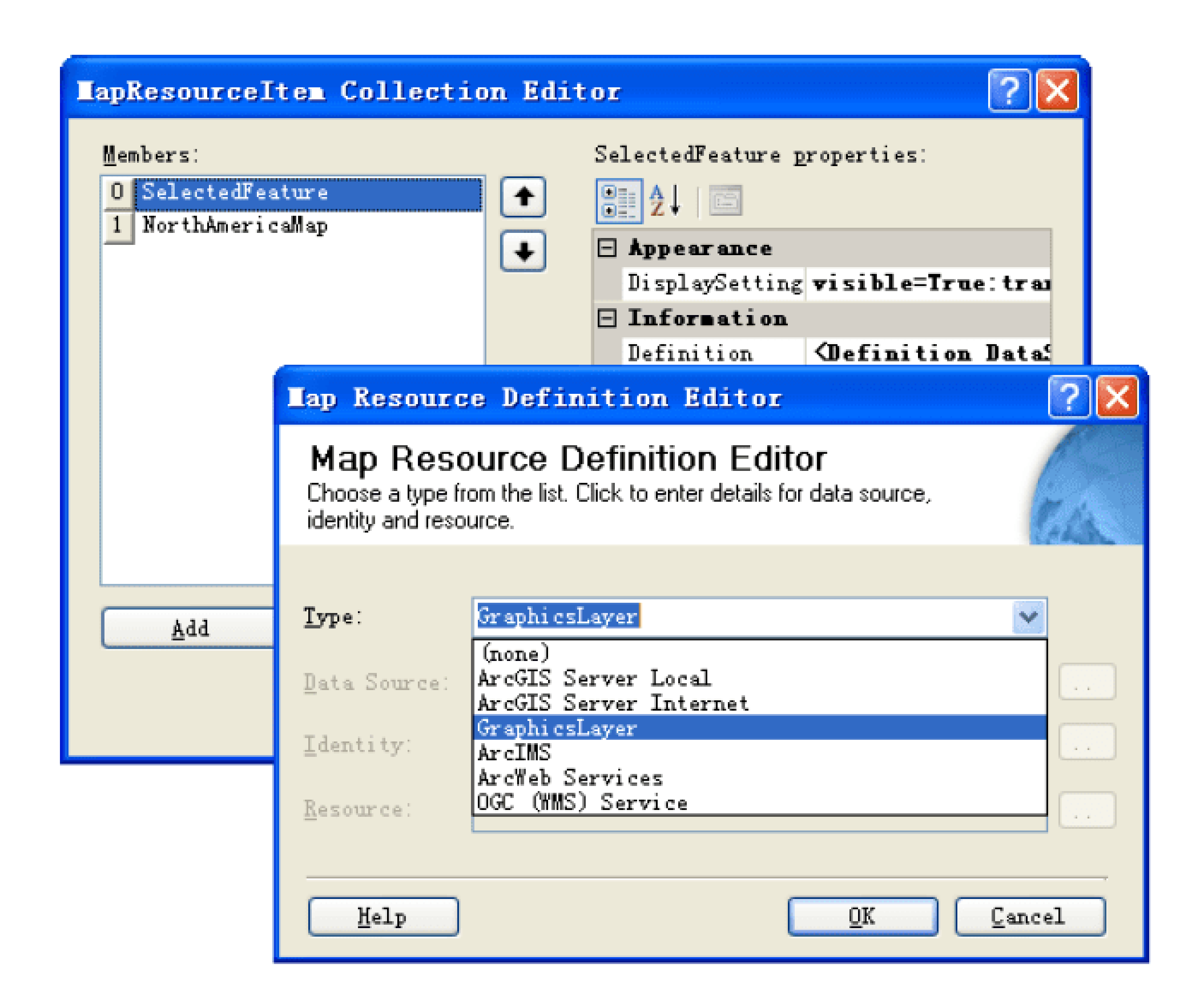


图 4.34　增加 GraphicsLayer 类型的资源

在 Default.aspx 页面的代码的 JavaScript 函数部分加入 AttributeQuery()函数的实现代码，代码如下：

```
function AttributeQuery(){
    var layer = document.getElementById("LayerList");
    var condition = document.getElementById("queryCondition");
    var message = 'ActiveType=AttributeQuery&EventArg=';
    message += layer.value;
    message += "&Condtion=" + condition.value;
    var context = 'Page1';
    <%=attributeQueryCallBack%>
}
```

切换到 Default.aspx.cs 文件中，在_Default 类中再加入如下公用字段：

```
public string attributeQueryCallBack;
```

然后在 Page_Load 方法中增加如下代码，生成调用的客户端脚本：

```
attributeQueryCallBack = Page.ClientScript.GetCallbackEventReference(
                         this, "message", "processCallbackResult",
                         "context", "postBackError", true);
```

然后修改 ICallbackEventHandler.GetCallbackResult()方法中针对不同 ActiveType 执行不同操作的 switch 部分代码，执行属性查询图形，代码如下：

```
// 是否进行缓存，对于属性查询图形不进行缓存
bool needCached = true;

// 针对参数中指定的 ActiveType 不同执行不同操作
string controlType = queryString["ActiveType"];
string eventArg = queryString["EventArg"];
switch (controlType)
```

```
    {
       case "GetResourceContent":
          response = GisFunctionality.GetResourceContent(Map1);
          break;
       case "ShowFieldInfo":
          response = GisFunctionality.ShowFieldInfo(Map1, eventArg);
          break;
       case "AttributeQuery":
          response    =    GisFunctionality.AttributeQuery(Map1,    eventArg,
queryString["Condtion"]);
          needCached = false;
          break;
       default:
          break;
    }

    if (needCached)
    {
       TimeSpan cacheDuration = new TimeSpan(0, 0, 999999);
       Cache.Add(callbackArg, response, null,
          DateTime.Now.Add(cacheDuration),
          System.Web.Caching.Cache.NoSlidingExpiration,
          System.Web.Caching.CacheItemPriority.NotRemovable, null);
    }
    return response;
```

这是由于每个用户每次查询的内容可能都不一样，因此没有必要将属性查询图形的结果保存下来。

GisFunctionality 类的 AttributeQuery 方法的代码如下：

```
    public static string AttributeQuery(Map map, string layerIndex,
    string condition)
    {
       IGISFunctionality gisfunc = map.GetFunctionality("NorthAmericaMap");
       if (gisfunc == null)
          return "";

       IGISResource gisresource = gisfunc.Resource;
       bool supportquery =
         gisresource.SupportsFunctionality(typeof(IQueryFunctionality));
       if (!supportquery)
          return "";

       IQueryFunctionality qfunc;
       qfunc = gisresource.CreateFunctionality(
               typeof(IQueryFunctionality), null)
               as IQueryFunctionality;

       SpatialFilter spatialfilter = new SpatialFilter();
       spatialfilter.ReturnADFGeometries = false;
```

```
        spatialfilter.MaxRecords = 1000;
        spatialfilter.WhereClause = condition;

        System.Data.DataTable datatable = qfunc.Query(
                                          null, layerIndex, spatialfilter);
        if (datatable == null)
            return "";

        return HighlightShow(map, datatable);
    }
```

在上面的代码中，先得到 NorthAmericaMap 的查询功能对象，然后利用此功能的 Query 方法，配合 SpatialFilter 对象，执行空间查询，得到查询结果，保存到数据表中。最后调用 HighlightShow 方法高亮显示满足条件的要素。

HighlightShow 方法的代码如下：

```
    private static string HighlightShow(Map map, DataTable datatable)
    {
        IGISFunctionality gisfunc = map.GetFunctionality("SelectedFeature");
        if (gisfunc == null)
            return "";

        ESRI.ArcGIS.ADF.Web.DataSources.Graphics.MapResource gResource = null;
        gResource = gisfunc.Resource as
          ESRI.ArcGIS.ADF.Web.DataSources.Graphics.MapResource;
        if (gResource == null)
            return "";

        // 从 SelectedFeature 资源中寻找 ElementGraphicsLayer
        ElementGraphicsLayer gLayer = null;
        foreach (System.Data.DataTable dt in gResource.Graphics.Tables) {
            if (dt is ElementGraphicsLayer)
            {
                gLayer = (ElementGraphicsLayer)dt;
            }
        }

        // 第一次调用时 ElementGraphicsLayer 不存在，需要创建一个
        if (gLayer == null) {
            gLayer = new ElementGraphicsLayer();
            gResource.Graphics.Tables.Add(gLayer);
        }

        //  清除已有数据
        gLayer.Clear();

        DataRowCollection drs = datatable.Rows;

        int shpind = -1;
        for (int i = 0; i < datatable.Columns.Count; i++) {
```

```
            if                      (datatable.Columns[i].DataType                    ==
typeof(ESRI.ArcGIS.ADF.Web.Geometry.Geometry)) {
                // 找到 Geometry 字段的序号
                shpind = i;
                break;
            }
        }

        try {
            foreach (DataRow dr in datatable.Rows) {
                ESRI.ArcGIS.ADF.Web.Geometry.Geometry geom = dr[shpind]
                    as ESRI.ArcGIS.ADF.Web.Geometry.Geometry;

                // 创建一个 GraphicElement
                GraphicElement ge = null;
                ge = new GraphicElement(geom, System.Drawing.Color.Yellow);
                ge.Symbol.Transparency = 50.0;

                //  将 GraphicElement 添加到 ElementGraphicsLayer 中
                gLayer.Add(ge);
            }
        }
        catch (InvalidCastException ice) {
            throw new Exception("No geometry available in datatable");
        }

        if (map.ImageBlendingMode == ImageBlendingMode.WebTier) {
            map.Refresh();
        }
        else if (map.ImageBlendingMode == ImageBlendingMode.Browser) {
            // 只刷新 Graphics Resource
            map.RefreshResource(gResource.Name);
        }

        return map.CallbackResults.ToString();
    }
```

编译并运行程序。在文本框中输入合适的条件，然后选择查询按钮，系统将高亮显示满足条件的要素，效果如图 4.35 所示。

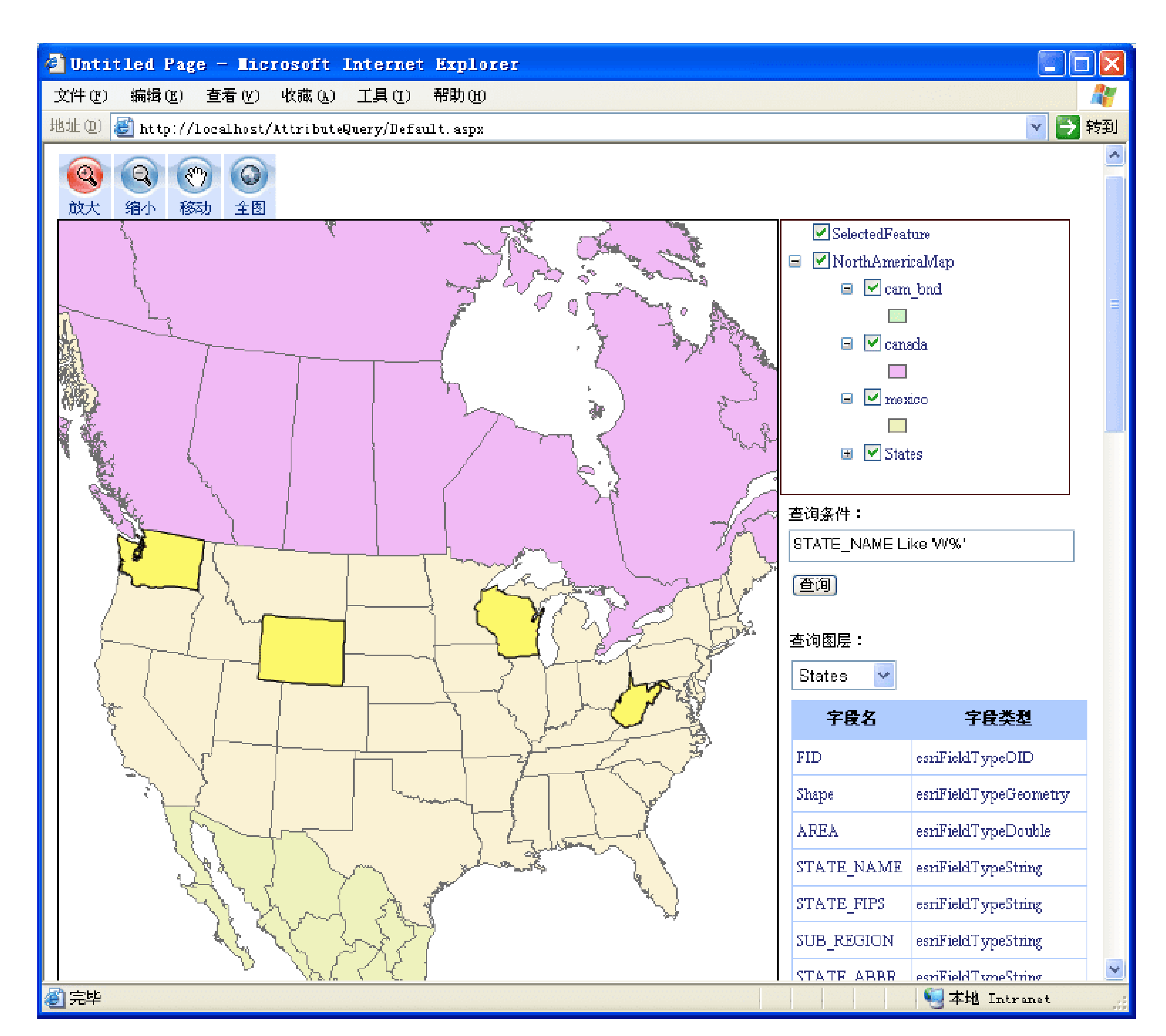


图 4.35　属性查询图形的效果

4.6　右键菜单与地图图片保存

由于地图控件自身就支持用鼠标滚轴来放大、缩小，用方向键来漫游以及用鼠标左键拖动来漫游，因此如果系统功能相对简单，就没有必要保留工具栏，可将一些不能通过鼠标与键盘直接实现的操作，放到右键菜单中，从而使地图尽可能占据大的空间。

Web ADF 提供有一个名为 ContextMenu 控件。通过该控件便可为应用程序提供右键菜单功能。每个 ContextMenu 控件可包含一个或多个菜单项，而每个菜单项与一段客户端或服务器端代码相连。此外，每个菜单项可以同时显示一个图表以及文字（也可只显示文字或图标）。要注意的是 Toc 控件与 TaskResult 控件自身就包含了右键菜单用于控制其中的节点。

本节将结合实现全图显示与将当前地图保存为图片功能演示如何使用右键菜单。为了更好地演示，在本实例中利用两个地图资源，一个是前面经常使用的 NorthAmericaMap，另一个是还没有发布的地图资源。因此请读者按照第 3 章介绍的方法将 C:\ProgramFiles\ArcGIS\ DeveloperKit\SamplesNET\Server\data\Usa\USA_Data.mxd 地图文档发布为名为 USAMap 的地图服务。

4.6.1　添加右键菜单

在 Visual Studio 中利用 File 菜单的 New Web Site 命令，创建一个名为 ContextMenu 的 Web 工程。

在 Default.aspx 页面中加入一个地图控件与一个地图资源管理器控件，分别命名为 Map1 与 MapResourceManager1。将地图控件的 MapResourceManager 属性指定为 MapResourceManager1，

从而使两个控件连接起来。

打开地图资源管理器控件的 ResourceItems 属性设置对话框，先后加入两个 ArcGIS Server Local 资源 USAMap 与 NorthAmericaMap，确保 USAMap 在 NorthAmericaMap 的上面。为了不使 USAMap 资源中的地图的背景覆盖 NorthAmericaMap 资源的地图，需要通过其 DisplaySetting 属性将资源的背景设置为透明。该属性设置对话框如图 4.36 所示，在该对话框中将透明背景颜色设置为白色，并选择 Make background transparent，即背景透明。

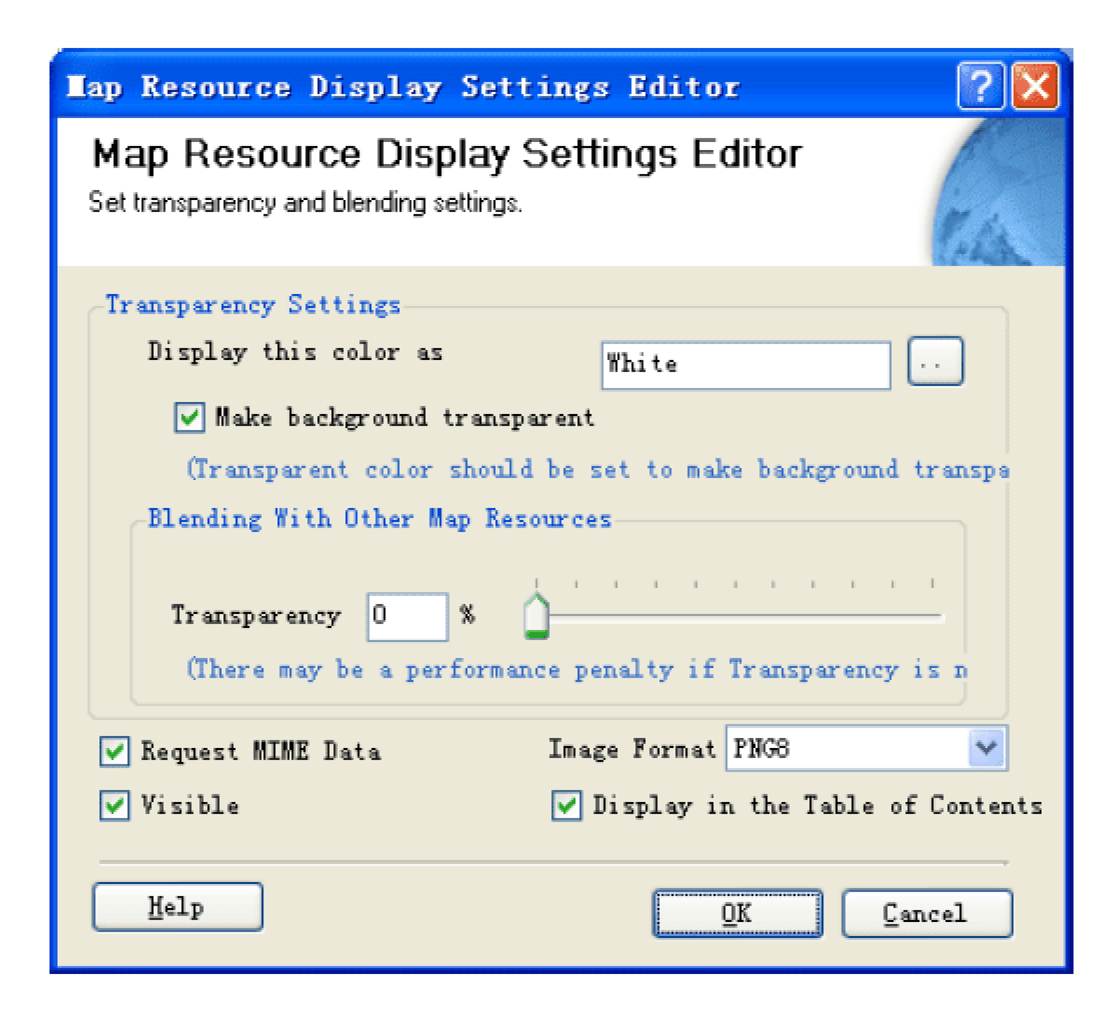


图 4.36 演示方式设置对话框

ContextMenu 控件没有集成到 Visual Studio 的工具箱中。因此，要使用该控件，可直接在页面中添加引用，或是先将该控件加入到 Visual Studio 的工具箱中，然后通过鼠标拖动加入到页面，推荐使用后者。

将 ContextMenu 控件加入到 Visual Studio 的工具箱中很容易，首先在工具箱中展开 ArcGIS Web Controls 选项卡，然后选择其右键菜单的 Choose Items…命令，打开如图 4.37 所示的 Choose Toolbox Items 对话框。在该对话框中定位并选择命名空间为 ESRI.ArcGIS.ADF.UI.WebControls 的 ContextMenu 控件。选择 OK 关闭对话框。此时该控件应该出现在工具箱中了。

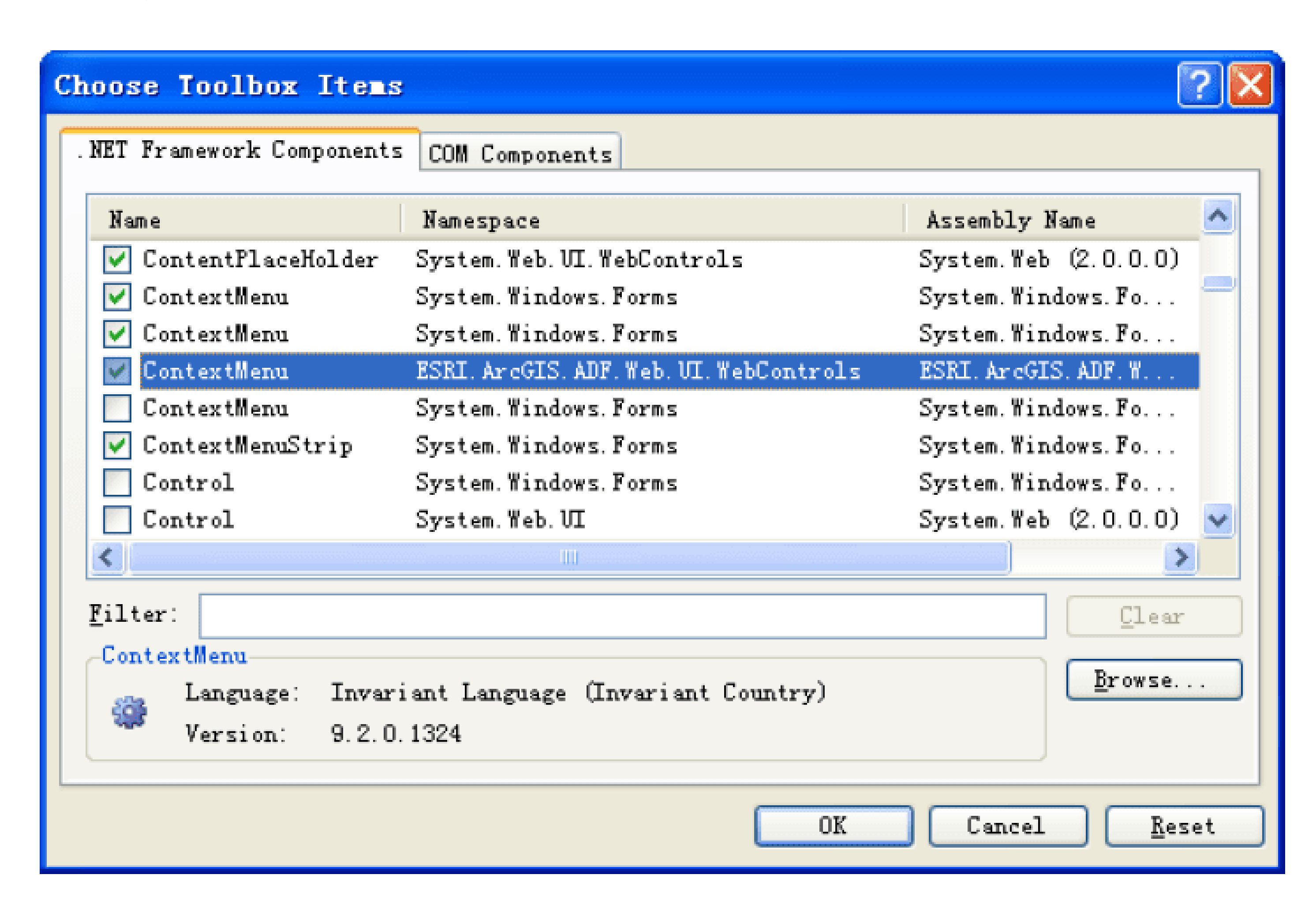


图 4.37 增加工具对话框

在 Default.aspx 中加入 ContextMenu 控件，命名为 ContextMenu1。注意 ContextMenu 控件在设计期间的位置并会不决定运行期间右键菜单的位置。

将 ContextMenu1 控件的 BackColor 属性设置为白色。该属性默认值为透明。

Web ADF 没有为 ContextMenu 控件在设计期间提供设计菜单项的对话框，因此需要通过代码来实现。可以用 ContextMenuItem 类来创建菜单项。ContextMenuItem 类的参数中包含一个图标的 url，菜单项的文字以及一代表当用户选择该菜单项所执行的 JavaScript 代码。最合适加入菜单项的时间应该是页面加载时。因此首先在 Page_Load 方法中加入如下代码，用于在菜单中添加两个菜单项：

```
if (IsPostBack)
    return;

ESRI.ArcGIS.ADF.Web.UI.WebControls.ContextMenuItem fullextMenuItem =
   new ESRI.ArcGIS.ADF.Web.UI.WebControls.ContextMenuItem(
   "images/fullextent.jpg", "全图", null);

ESRI.ArcGIS.ADF.Web.UI.WebControls.ContextMenuItem saveMenuItem =
   new ESRI.ArcGIS.ADF.Web.UI.WebControls.ContextMenuItem(
   "images/save.jpg", "另存为图片", null);

ContextMenu1.Items.Add(fullextMenuItem);
ContextMenu1.Items.Add(saveMenuItem);
```

为了显示右键菜单，需要在合适的控件上增加事件。本实例中，希望当用户右键单击地图时显示右键菜单。在运行期间，地图控件在浏览器中创建了一个 Div 来显示地图数据，因此可使用该 Div 的属性来设置。其中 oncontextmenu 属性使浏览器监听右键单击事件。当事件触发时，使用 JavaScript 代码来显示该右键菜单及其菜单项。Web ADF 提供了一系列的 JavaScript 函数来显示与隐藏右键菜单。在本实例中，当地图 oncontextmenu 事件触发时，JavaScript 函数 esriShowContextMenu()将被调用，并显示右键菜单。在 Page_Load 方法中再加入如下代码：

```
string format = "esriShowContextMenu(event,'{0}','{1}','{2}');return false;";
string showContextMenu = string.Format(format, ContextMenu1.ClientID,
                                 Map1.UniqueID, "");

Map1.Attributes.Add("oncontextmenu", showContextMenu);
```

4.6.2 处理右键菜单事件

在程序运行期间一旦显示了右键菜单，用户便应可以选择某菜单项来启动一个动作，此时触发的是 ContextMenu 控件的 ItemClicked 事件。当一菜单项被选择时，一个回调请求就会发送到包含菜单的页面。可以通过菜单项的 Text 属性来判断用户选择的是哪个菜单项。要注意的是，ContextMenu 控件生成的回调响应规定由 Web ADF JavaScript 来处理，因此，如果页面中其他控件的内容改变必须打包成 CallbackResults，并加入到 ContextMenu 的 CallbackResults 属性中。

在属性设置面板中，双击 ContextMenu1 的 ItemCliked 事件，生成事件响应方法 ContextMenu1_ItemClicked。在其中加入如下代码（另存为图片功能在下一小节中实现）：

```
switch (args.Item.Text) {
    case "全图": {
            MapHelper.ShowFullEextent(Map1);
            ContextMenu1.CallbackResults.CopyFrom(Map1.CallbackResults);
            break;
        }
}
```

在上述代码中调用了 MapHelper 类的 ShowFullEextent 方法实现显示全图范围。

在工程中加入 App_Code 目录，然后在其中加入 C#类 MapHelper。在 MapHelper.cs 的头部加入如下命名空间的引用：

```
using ESRI.ArcGIS.ADF.Web.UI.WebControls;
using ESRI.ArcGIS.ADF.Web.DataSources;
using ESRI.ArcGIS.ADF.ArcGISServer;
using ESRI.ArcGIS.ADF.Web.DataSources.ArcGISServer;
```

ShowFullEextent 方法的代码如下：

```
public static void ShowFullEextent(Map map)
{
    ESRI.ArcGIS.ADF.Web.Geometry.Envelope fullExtent = null;
    foreach (IMapFunctionality imf in map.GetFunctionalities()) {
        MapFunctionality mf = imf as MapFunctionality;
        ESRI.ArcGIS.ADF.ArcGISServer.MapDescription mapDescription =
           mf.MapDescription;
        ESRI.ArcGIS.ADF.ArcGISServer.Envelope envServer =
           mapDescription.MapArea.Extent;
        ESRI.ArcGIS.ADF.Web.Geometry.Envelope env = null;
        env =  ESRI.ArcGIS.ADF.Web.DataSources.
               ArcGISServer.Converter.ToAdfGeometry(envServer)
               as ESRI.ArcGIS.ADF.Web.Geometry.Envelope;
        if (fullExtent == null) {
            fullExtent = env;
        }
        else {
            fullExtent.Union(env);
        }
    }

    map.Extent = fullExtent;
    map.Refresh();
}
```

在上述代码中，我们对地图中的所有绘图功能进行循环，当然对于本实例分别是 USAMap 与 NorthAmericaMap，得到每个绘图功能的地图描述对象（MapDescription），通过该对象的 Map.Area.Extent 属性，得到该地图的坐标范围，然后将这些范围进行 Union 操作，得到整个地图的坐标返回，最后将该坐标返回设置为地图当前显示返回，即实现全图显示。

编译并运行程序。先用鼠标滚轴放大或缩小地图，然后通过右键菜单的“全图”命令即可返回。程序运行界面如图 4.38 所示。

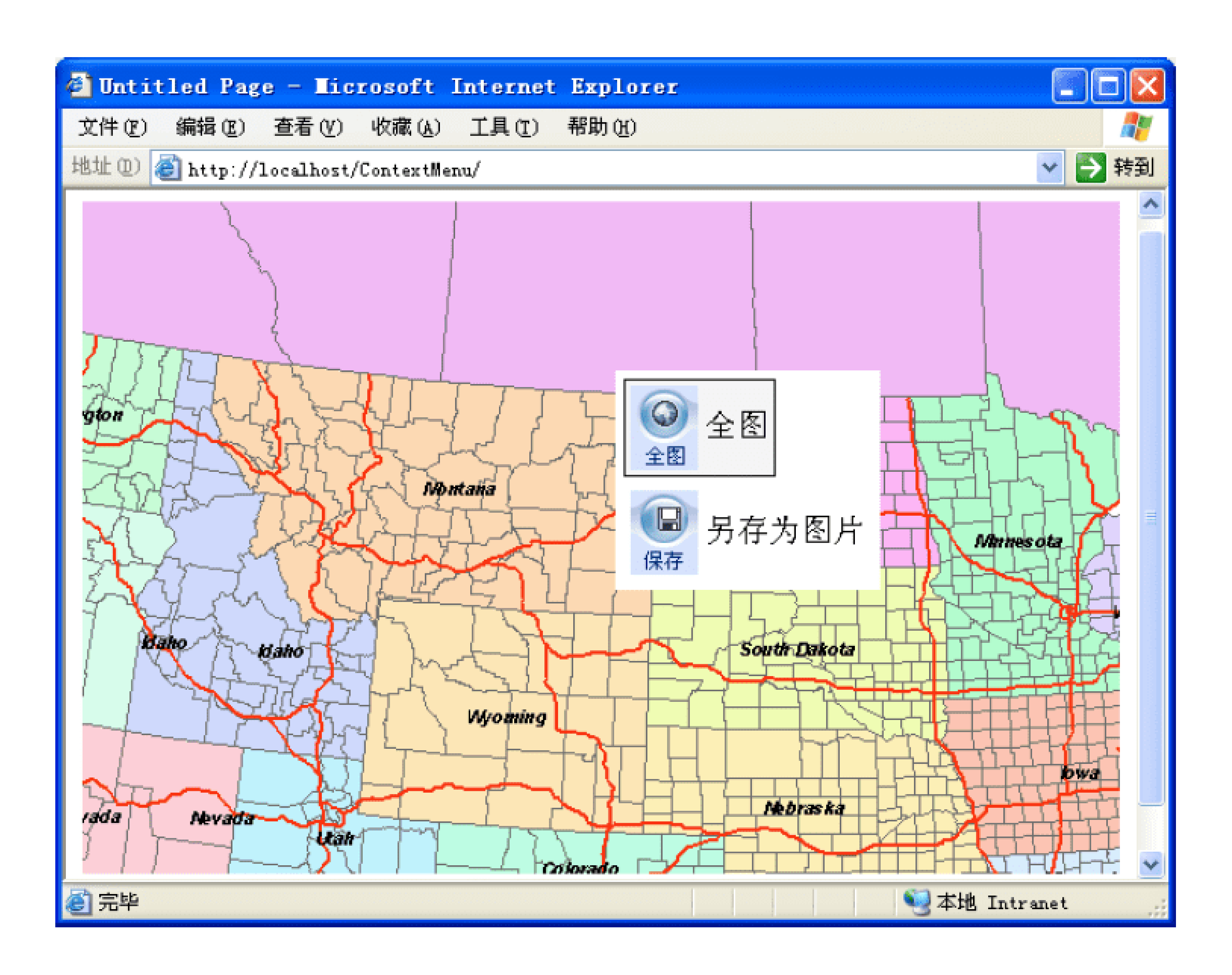


图 4.38 程序运行效果

4.6.3 保存地图图片

下面要实现的是右键菜单的图片保存功能。

对于包括多个资源的地图控件（本实例就如此），将生成多张图片。这些地图图片在浏览器端或是在 Web 服务器端融合，尽管用户看似是一张完整的地图，实际上包含了许多叠加图片。这时，需要将所有图片融合成一张图片在地图控件之外使用。生成一张可打印的图片。下面将演示如何使用.NET 的 GDI+来生成一张融合好的图片。

在 ContextMenu1_ItemClicked 方法的 switch 块中加入如下代码，用于响应“另存为图片”菜单项：

```
case "另存为图片": {
        MapHelper.SaveAsPicture(Map1);
        ContextMenu1.CallbackResults.CopyFrom(Map1.CallbackResults);
        break;
    }
```

MapHelper 类的 SaveAsPicture 方法的代码如下：

```
public static void SaveAsPicture(Map map)
{
    int imgHeight = (int)map.Height.Value;
    int imgWidth = (int)map.Width.Value;
    System.Collections.Generic.IList<Bitmap> bitmaps =
       new System.Collections.Generic.List<Bitmap>();

    foreach (IMapFunctionality mf in map.GetFunctionalities()) {
       ESRI.ArcGIS.ADF.Web.MapImage mi = mf.DrawExtent(map.Extent);
```

```
            if (mi != null)
                bitmaps.Add(mi.GetImage());
        }

        Bitmap newImg = new Bitmap(imgWidth, imgHeight);
        Graphics imgGraphics = Graphics.FromImage(newImg);
        imgGraphics.FillRectangle(new SolidBrush(System.Drawing.Color.LightGray),
0, 0, imgWidth, imgHeight);

        for (int j = bitmaps.Count - 1; j >= 0; j--) {
            Bitmap bmp = bitmaps[j];
            imgGraphics.DrawImage(bmp, 0, 0, imgWidth, imgHeight);
        }
        string fileNameVirtual = System.DateTime.Now.ToString() + ".jpg";
        fileNameVirtual = fileNameVirtual.Replace(":", "-");
        string fileName = map.Page.MapPath("Output\\" + fileNameVirtual);
        newImg.Save(fileName, System.Drawing.Imaging.ImageFormat.Jpeg);
        imgGraphics.Dispose();
        newImg.Dispose();

        string functionValue = "open('SavePicture.aspx?FileName=" +
                               fileNameVirtual + "', 'SavePicture');";
        object[] oa = new object[1];
        oa[0] = functionValue;
        CallbackResult cr = new CallbackResult(
                            null, null, "javascript", functionValue);
        map.CallbackResults.Add(cr);
    }
```

在上述代码中，先对地图中的所有绘图功能进行循环，将所生成的图片加入到一个列表中，接着利用 GDI+方法将列表中的图片绘制成一个图片，然后将该图片保存在 Output 目录下（需要读者在工程中加入该目录），最后构造 JavaScript 代码，新打开一窗口，以图片的文件名为参数显示 SavePicture.aspx 页面。我们将在该页面中实现显示或保存图片操作。

SavePicture.aspx 是一虚拟的页面，对应的是一个实现了 System.Web.IHttpHandler 接口的类。

切换到 Web.Config 文件中，<system.web>与</system.web>之间加入如下代码，声明地址为 SavePicture.aspx 的访问，由 DownloadMapHandler 类来处理：

```
<httpHandlers>
  <add verb="*" path="SavePicture.aspx" type="DownloadMapHandler"/>
</httpHandlers>
```

在 App_Code 目录中加入 C#类 DownloadMapHandler。该类的代码如下：

```
public class DownloadMapHandler : System.Web.IHttpHandler {
    #region IHttpHandler Members

    bool IHttpHandler.IsReusable {
        get {
            return true;
```

```
        }
    }

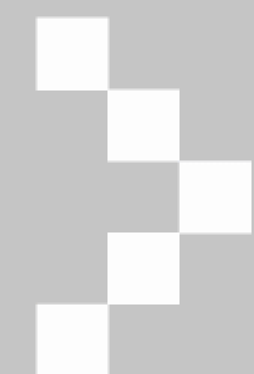
    void IHttpHandler.ProcessRequest(HttpContext context) {
        NameValueCollection parms = context.Request.Params;
        string fileName = parms["FileName"].ToString();
        fileName = "Output\\" + fileName;
        string path = context.Server.MapPath(fileName);
        context.Response.Clear();
        context.Response.BufferOutput = true;
        context.Response.ContentType = "application/octet-stream";
        context.Response.AppendHeader("Content-Disposition",
"attachment; filename=" + fileName);
        context.Response.TransmitFile(path);

        // 将输出发送到客户端
        context.Response.Flush();
        context.Response.End();
    }

    #endregion
}
```

编译并运行程序，检查并执行保存图片功能。

第 5 章



数据源、资源与功能对象

在第 4 章中我们主要使用的数据源是 ArcGIS Server 的本地服务，但是正如前面经常提到的，Web ADF 为了更好的支持多数据源，在 COM 对象的基础上，增加了 ADF 9.0/9.1 没有的公有 API、值对象等开发层次。同时，提供了更好的控制粒度。在值对象以及 COM 对象之间，可以根据需要（如提高性能等），按照一定的分层设计规范的使用。在这一章中我们将介绍这些不同的数据源及其对应的资源与功能对象的使用。

通过本章你将了解到：

5.1 数据源

Web ADF 控件可以使用多种来源的数据，包括本地或远程连接的 ArcGIS Server 服务、ArcIMS 服务、ArcWeb 服务、WMS 服务以及 Web ADF 的图形数据。Web ADF 的一个主要优势是能同时显示一个或多个数据源生成的多个地图，并通过地图控件来显示，而不需要开发人员编写任何代码。在 Web ADF 中，将数据源定义为资源，数据源的能力决定了资源的能力及其所能提供的功能。

Web ADF 中使用的任何数据源实质上是一自定义的数据源，它们通过实现 Web ADF 的公有数据源 API 接口，插入到 Web ADF 中。基本上每种数据源都有一个特有 API，只能供该数据源使用。要在 Web ADF 中将某数据源作为资源来使用，那么必须实现相应的公有 API 的接口与类。数据源的功能决定了需要实现哪些接口与类。例如 ESRI.ArcGIS.ADF.ArcGISServer 程序集包含与 ArcGIS Server SOAP API 关联的 SOAP 代理对象与值对象，除了提供自身使用之外，还可以供 Web ADF 控件与 API 之外的其他类来访问 ArcGIS Server 服务。为了在 Web ADF 中将 ArcGIS Server 的服务作为一个数据源，ArcGIS Server SOAP API 提供了 ESRI.ArcGIS.ADF.Web.DataSources.ArcGISServer 程序集，该程序集实现了公有 API 的接口。Web ADF 就是利用该程序集来使用本地与远程 ArcGIS Server 数据源的。

Web ADF 中每个数据源都有自己的类库，都包含在 ESRI.ArcGIS.ADF.Web.DataSources 命名空间中。数据源的类库包含了资源与功能类，通过这些类访问数据源的基本能力。例如，如果一个数据源能生成一张地图，那么它就支持地图资源（MapResource）与绘图功能（MapFunctionality）。地图控件以及开发人员利用地图资源创建绘图功能，并由绘图功能生成地图。

5.1.1 ArcGIS Server 本地数据源

一个 ArcGIS Server 本地数据源代表在局域网中，通过服务器对象管理器（SOM）连接一个 ArcGIS Server 服务。为了连接服务，必须指定服务器的名称（或 IP 地址）与服务的名称，此外还需要提供身份信息，该身份需要是装有 SOM 计算机中 agsusers 或 agsadmin 用户组中的成员。

ArcGIS Server 本地数据源可代表 ArcGIS Server 地图服务、地理编码服务以及空间处理服务。本地数据源允许开发人员通过服务器上下文对象来操作细粒度的 ArcObjects，因此可以在 Web ADF 应用程序中利用 ArcObjects 的强大功能，例如编辑要素图层、网络分析以及投影转换等。

在程序中可以通过 ArcGIS Server 数据源类库（ESRI.ArcGIS.ADF.Web.DataSource.ArcGISServer），来连接与操作 ArcGIS Server 本地数据源。通过该程序集中的 MapFunctionality 类，可以访问地图描述信息，甚至使用 ArcGIS Server SOAP API。ArcGIS Server 本地资源，例如 MapResourceLocal，同时提供了通过 ServerContextInfo.ServerContext 属性来访问服务器的上下文对象。通常，与 ArcGIS Server 本地数据源打交道的资源类的类名以“Local”结束，例如 Web ADF 中的 MapResourceLocal 类。

5.1.2　ArcGIS Server 远程数据源

选择 ArcGIS Server 远程数据源意味着客户端将通过 Web 服务端点来访问 ArcGIS Server 服务。ArcGIS Server 服务提供一 WSDL（Web Service Description Language，Web 服务描述语言），客户端可利用该 WSDL 在开发环境中生成 ArcGIS Server SOPA API 相应的代理类。对于 ArcGIS Server 服务，SOAP API 只提供了 ArcObjects 的部分功能，即无状态那部分功能，因此 ArcGIS Server 远程数据源只能与 ArcGIS Server 服务使用无状态方式交互。同时由于不能通过 ArcGIS Server 远程数据源得到服务器上下文，因此细粒度的 ArcObjects 功能也不能使用。

ArcGIS Server 远程数据源可以是地图服务、地理编码服务以及空间处理服务。在地图资源管理器中，当设置 ArcGIS Server 远程连接时，应提供包含服务的服务器的 URL，格式一般是 http://serverIP/arcgis/services，然后输入认证信息，最后从服务下拉列表框中选择一个服务，如图 5.1 所示。

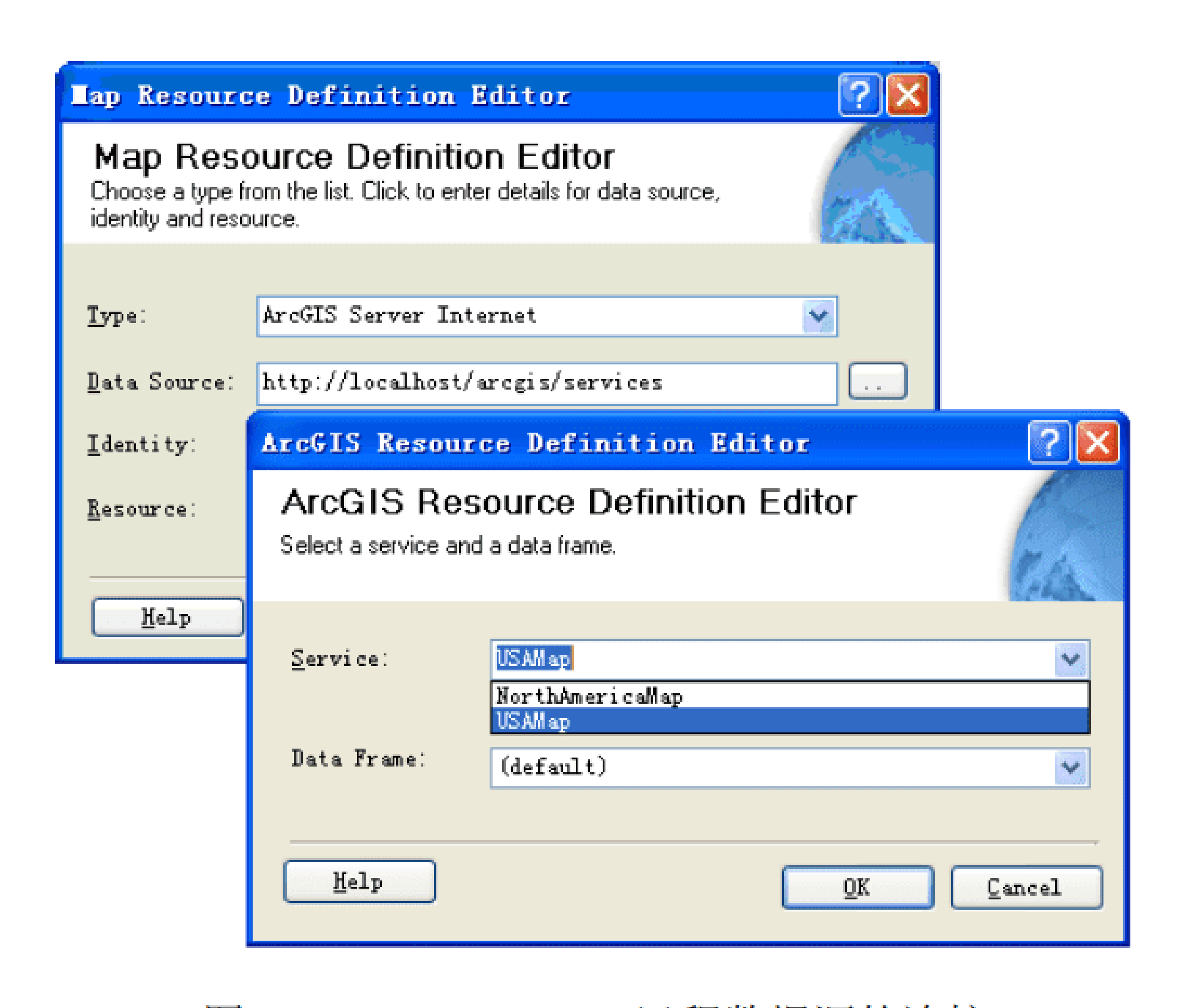


图 5.1　ArcGIS Server 远程数据源的连接

ArcGIS Server 远程数据源必须是池化的 GIS 服务。池化的服务可以让 Web 应用程序在用户之间共享。非池化的 GIS 服务的实例为单独的用户占有，当用户操作完成后，该实例被销毁。如果在远程连接中使用非池化的服务，那么每个地图或其他 GIS 的操作都会创建与销毁一个新的进程，这将导致 GIS 服务器资源的低效率使用。

可以通过 ArcGIS Server 数据源类库（ESRI.ArcGIS.ADF.Web.DataSource. ArcGISServer），来连接与操作 ArcGIS Server 远程数据源。通常与 ArcGIS Server 远程数据源打交道的类的名称以“Internet”结束，例如 MapResourceInternet。

5.1.3　ArcIMS

若在地图资源管理器设置地图资源的对话框中选择 ArcIMS 数据源，还需要进一步设置使用

TCP 或 HTTP 来访问，如图 5.2 所示。对于 TCP 连接，需要指定 ArcIMS 应用服务器的 IP 地址与端口号（默认为 5300）。对于 HTTP 连接，需要指定 ArcIMS 的 servlet 连接的 URL。

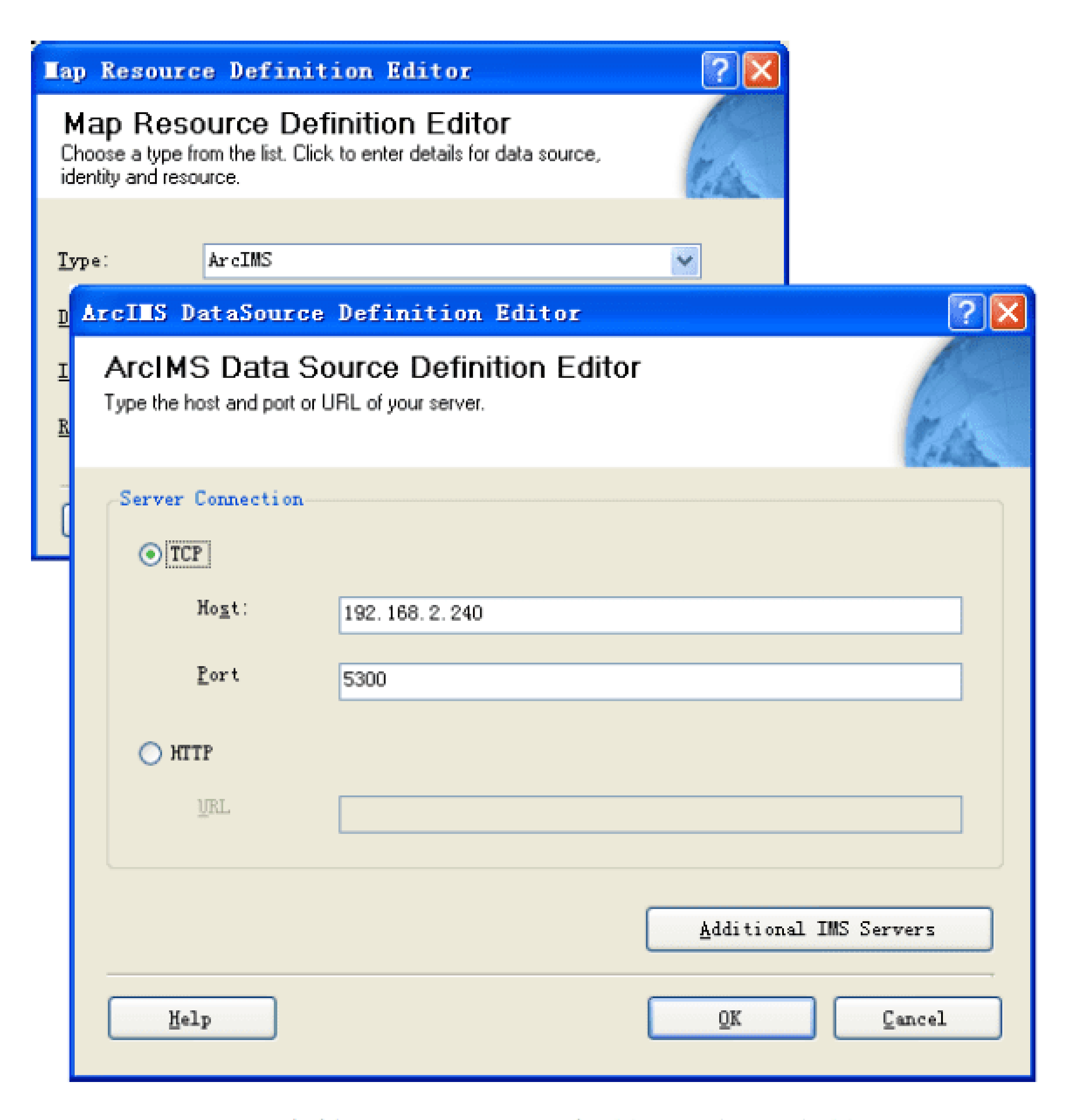


图 5.2　连接 ArcIMS 服务的两种方式的设置

ArcIMS 数据源在程序中的访问是使用 ArcIMS 数据源类库 ESRI.ArcGIS.ADF.Web.DataSource.IMS 来进行。通过该类库中的 MapFunctionality 类，可以访问 MapView 类。同时，还可以通过 ArcIMS 特有的 API 通过 ArcXML 访问更细粒度的功能。

5.1.4　图形图层

图形图层用于在 Web 服务器端快速显示地理要素。这些要素可以是也可以不是来源于 ArcGIS Server 服务、WMS 服务或 ArcIMS 服务。例如，可以对地图中已有的服务进行地理编码操作，返回一点图形数据表。

可以在地图资源管理器中通过加入一类型为“GraphicsLayer”的地图资源，作为图形数据源。图形资源的类型是 System.Data.DataSet，即数据集，因此可以包含许多数据表。通过图形资源的 Graphics 属性来访问 Web ADF 的 GraphicsDataSet 类。

Web ADF 中有两类图形图层，分别是 ElementGraphicsLayer 与 FeatureGraphicsLayer，都属于 System.Data.DataTable 类型，因此可以加入到 GraphicsDataSet 数据表集合中。图形图层的内容存储在应用程序的内存中，因此，图形图层中内容的大小与应用程序需要的内存成正比。还需要注意的是，只能通过程序代码来创建与管理图形图层。

- ElementGraphicsLayer 类用来存储基本的图形元素，即几何图形与符号。一个 ElementGraphicsLayer 可以存储不同类型的几何图形。通常，不用来存储属性，常用来显示地图中选择的要素；
- FeatureGraphicsLayer 类用来仿真一真实的要素图层。每个 FeatureGraphicsLayer 只支持一

类几何图形。Web ADF 可以依据数据表中的属性来符号化几何图形。FeatureGraphicsLayer 同时还支持查询。

Web ADF 在图形图层资源中绘制每个图形图层，因此，Web ADF 同时也提供了一系列的几何类型、符号。

5.2　公有数据源 API

在 4.2 节中我们简单介绍了公有数据源 API（或简称为公有 API）的基本结构，但并没有深入说明，因此在本节中再详细介绍。

5.2.1　公有数据源 API 的内容

Web ADF 使用公有 API 来处理多种数据源。公有 API 是 Web ADF 中最重要的部分，因此它为在 Web ADF 中使用数据源提供了一抽象的框架。公有 API 的最基础的三个接口分别是：IGISDataSource（数据源接口）、IGISResource（资源接口）与 IGISFunctionality（功能接口）。

通常，在 Web ADF 中，代码是通过资源与功能对外提供的。一个数据源可以有一个或多个资源，一个资源可以有一个或多个功能。数据源的类型决定了可供使用的资源的类型，而数据源的能力决定了通过资源可访问的功能的类型。公有 API 提供了一组标准的资源与功能接口，数据源需要实现这些接口。表 5-1 列出了 Web ADF 中的资源接口及其说明。资源接口都继承于 IGISResource。

表 5-1　Web ADF 中的资源接口及其说明

资源	说明
IMapResource	根据要素或影像数据绘制地图的数据源
IGeocodeResource	支持地址匹配的数据源
IGeoprocessingResource	支持空间处理分析的数据源

功能接口都继承于 IGISFunctionality，每一个功能只能与一个类型的资源连接。Web ADF 的功能接口如表 5-2 所示。

表 5-2　Web ADF 中的功能接口及其描述

功能接口		说明
地图资源	IMapFunctionality	绘制地图
	IQueryFunctionality	空间与属性查询
	IMapTocFunctionality	为 Toc 控件提供信息
	ITileFunctionality	通过事先生成的地图切片提供地图数据
	IScalebarFunctionality	为地图比例尺提供信息
地理编码资源	IGeocodeFunctionaltiy	支持地址匹配
空间处理资源	IGeoprocessingFunctionality	支持空间处理操作

为了在 Web ADF 中使用自定义的数据源，就必须实现数据源接口、资源接口与功能接口。除了这些接口外，Web ADF 还提供了一组使用数据源的类库，这些数据源包括 ArcGIS Server、ArcIMS、ArcWeb、OGC\WMS 与 Web ADF 图形。图 5.3 描述了公有 API 抽象框架、一组 ArcGIS Server 与 ArcGIS 的实现，以及 Web ADF 控件与地图资源管理器是如何通过公有 API 访问数据源的。

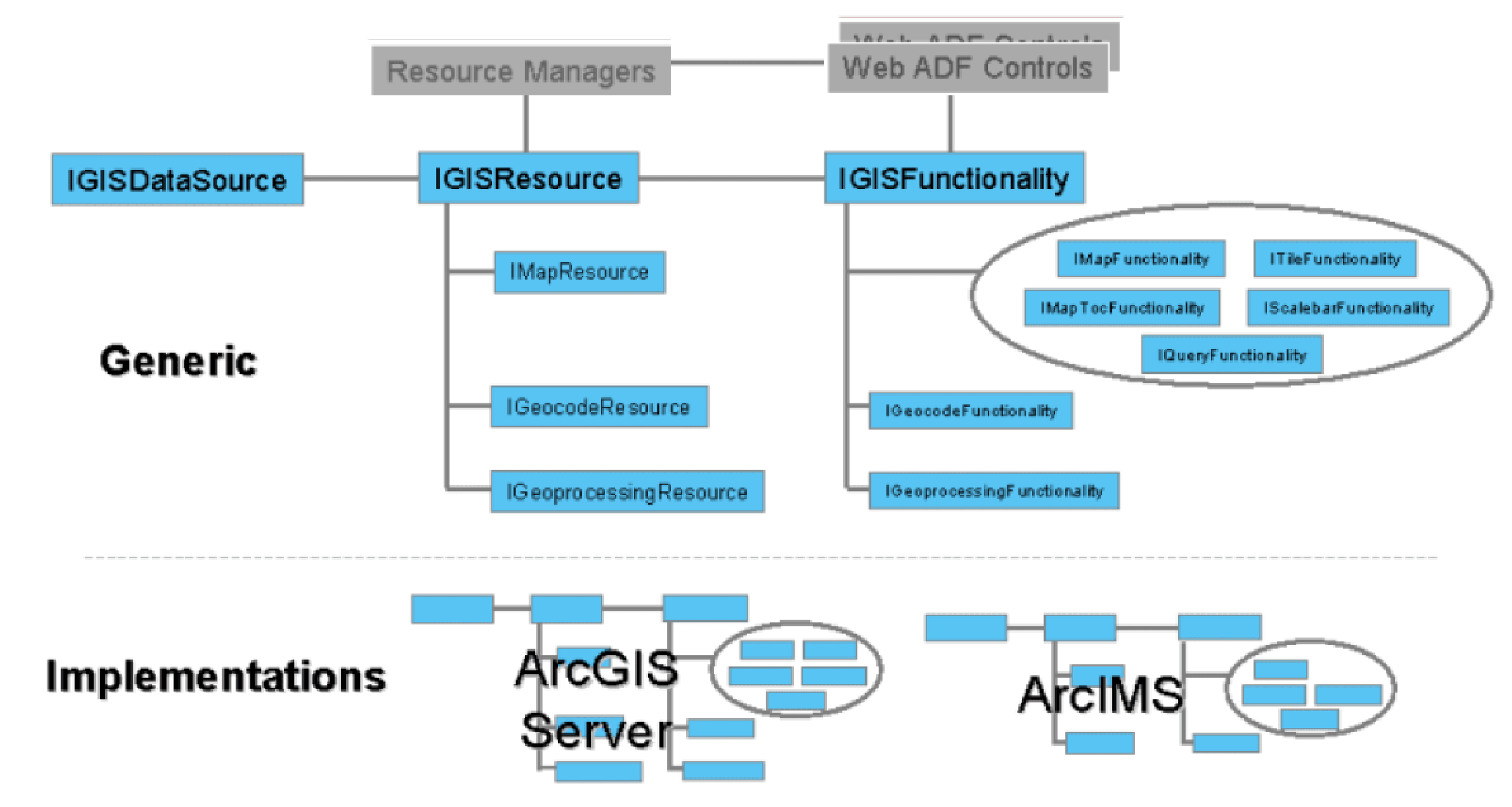


图 5.3 公有 API 的内容及其相互关系

一些数据源，例如 ArcGIS Server 与 ArcIMS，除了使用上述公有 API 之外，为了能更细粒度地访问数据源，还有专有 API。表 5-3 列出了数据源类型及其对应的公用 API 实现类库以及专有 API 类库。

表 5-3 数据源类型及其对应的公有 API、专有 API 类库

数据源	公有 API 类库	专有 API 类库
ArcGIS	ESRI.ArcGIS.ADF.Web.DataSources.ArcGISServer.dll	ESRI.ArcGIS.ADF.ArcGISServer.dll
ArcIMS	ESRI.ArcGIS.ADF.Web.DataSources.IMS.dll	ESRI.ArcGIS.ADF.IMS.dll
ArcWeb	ESRI.ArcGIS.ADF.Web.DataSources.ArcWebService.dll	ESRI.ArcGIS.ADF.ArcWebService.dll
OGC\WMS	ESRI.ArcGIS.ADF.Web.DataSources.OGCWMSService .dll	无
Web	ESRI.ArcGIS.ADF.Web.DataSources.Graphics.dll	ESRI.ArcGIS.ADF.Web.dll

公有 API 的一个最大优势是可以同时处理不同的数据源，通过其中的接口我们可以不理会具体使用什么方式访问数据源的，也不需要了解数据源是 ArcIMS、ArcGIS Server 还是 ArcWeb 服务，而直接使用资源与功能即可。

5.2.2 公有数据源 API 的实现

下面我们通过本地连接 ArcGIS Server 数据源来介绍公有 API 的实现。

如图 5.4 所示，IGISDataSource 接口定义与特定数据源的连接，以及连接时使用的身份以及连接状态。实现该接口的数据源需要提供访问数据源的连接对象与参数。在图 5.4 中，在 ArcGIS Server API 中的对象也实现了公有 API 中的 IGISDataSource 接口。虽然两个实现都维护公用的属性，例如身份与状态信息，但是依据数据源类型，它们定义不同连接属性。ArcGIS Server 本地连接要求使用连接库（ESRI.ArcGIS.ADF.dll 与 ESRI.ArcGIS.ADF.Connection.dll），而 ArcGIS Server 远程数据源不需要连接库，只要一个表示 Web 服务端点的 URL。因此就创建了使用本地连接的特定的 IGISDataSource，即 GISDataSouceLocal。

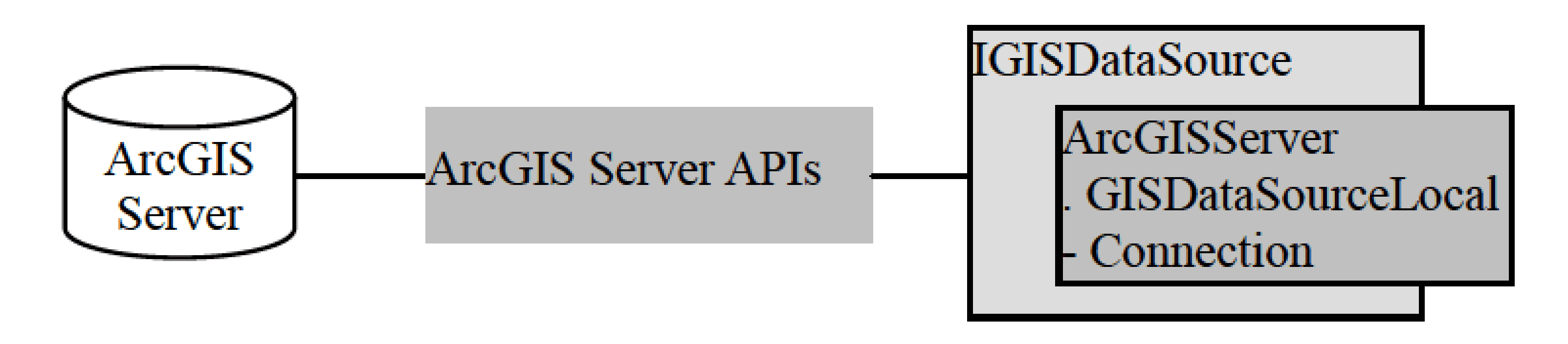


图 5.4　GISDataSourceLocal 的创建

通常，数据源决定了资源类型。每个数据源可以有多个资源。IGISResource 是资源的最顶层的接口，是 IMapResource 与 IGeocodeResource 两个常用的资源接口。IMapResource 代表数据源可为地图提供空间数据、绘制地图以及查询。IGeocodeResource 代表数据源可用于地址匹配。

如图 5.5 所示，ArcGIS Server 实现了创建 MapResouce 与 GeocodeResource 的接口。由于是 ArcGIS Server 本地数据源，该 MapResource 接口同时提供访问地图服务器对象与服务器上下文。同样，GeocodeResource 也提供了访问地址匹配服务器对象与服务器上下文。

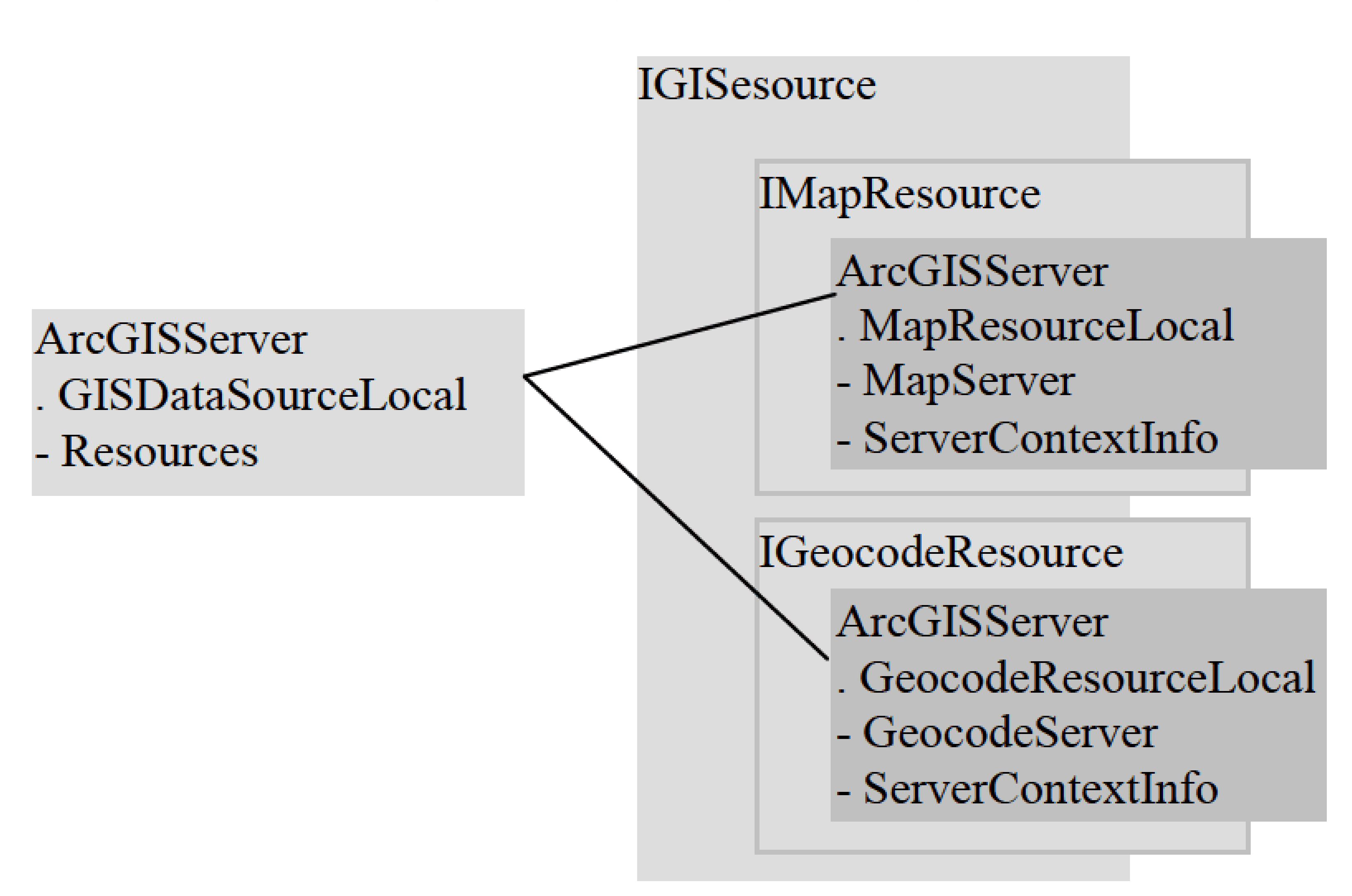


图 5.5　ArcGIS Server 本地数据源的资源

每个资源又可以有多个功能。如果一个资源支持某功能，那么该资源就能创建该功能，然后利用该功能执行操作。IGISFunctionality 是功能中最顶层的接口，是 IMapFunctionality 与 IQueryFunctionality 两常用的功能接口。由 MapResource 来创建这两功能。IMapFunctionality 表示资源的绘图能力，例如生成地图图片、设置或得到显示范围以及改变图层的可见性等。IQueryFunctionality 表示资源的空间与属性查询能力，例如使用一个 where 语句以及图形来返回要素的几何图形与属性。

如图 5.6 所示，ArcGIS Server 实现了创建 MapFunctionality 与 QueryFunctionality 的接口。虽然 MapDescription 对象专属于 ArcGIS Server 实现 IMapFunctionality 的类，不过功能的大多数方法与属性都可以通过上层抽象接口来调用。

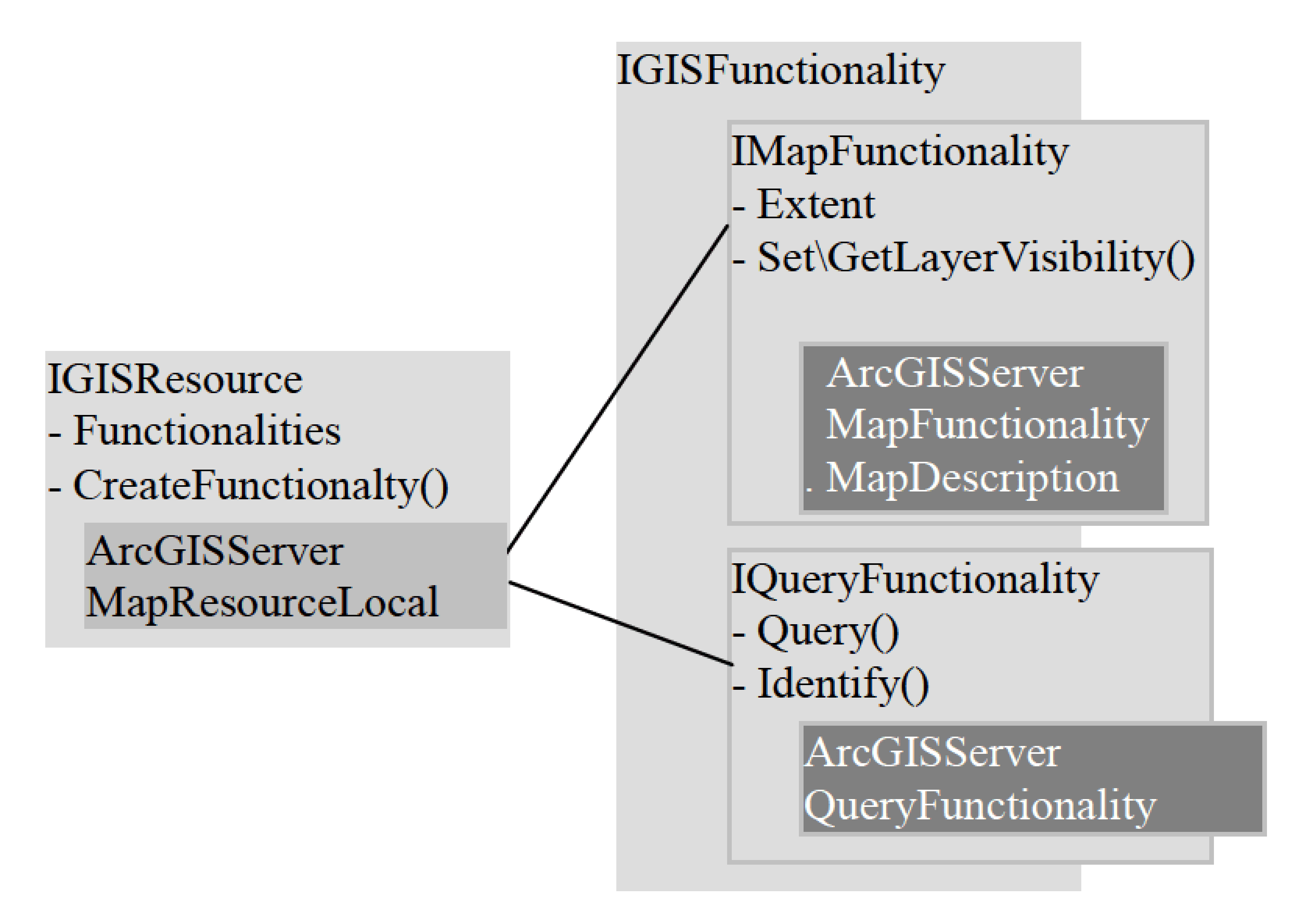


图 5.6 ArcGIS Server 本地地图资源的功能

5.3 ArcGIS Server 数据源的使用

在 5.1 节中我们泛泛地介绍了 Web ADF 自身提供的类库可以访问的数据源类型，但是并没有深入介绍每个数据源使用的方式。通常，使用最广泛的是 ArcGIS Server 数据源，因此有必要深入介绍下 ArcGIS Server 数据源的使用。

5.3.1 ArcGIS Server API

程序员可以通过两类 API 来访问与操作 ArcGIS Server 服务，分别是 ArcGIS Server ArcObjects API 与 ArcGIS Server SOAP API。

ArcObjects API 使用客户端的 ArcObjects 互操作程序集与对象库和 GIS 服务中的 ArcObjects 打交道，因此只有直接通过 SOM 访问 ArcGIS 服务器时才能使用该方式。该 API 包含了 ArcObjects 类库才能提供的深入的实用功能。

SOAP API 是与基于 SOAP 标准的 ArcGIS Server 服务通信的 XML 结构的语言。服务器对象以及一些服务器扩展对象定义了一组 SOAP 标签与属性，用于处理 SOAP 请求与生成 SOAP 响应。由于该功能在服务器对象层面上，开发人员可以直接使用 SOAP，而不需要使用 Web 服务，便可与服务器对象交互。ArcGIS Server 的 Web 服务通过 Web 服务端点对外提供该功能，但是 SOAP 请求及响应仍然直接由服务器对象或扩展对象处理。使用 SOAP API 最简单的方法是在开发环境中

使用 SOAP 工具包生成代理类。一般不直接操作 SOAP 字符串，而是使用本地代理类与值对象，通过 Web 服务端点或 SOM，与 ArcGIS Server 服务交互。SOAP API 只能保持很有限的状态信息，因此需要客户端应用程序负责维护状态。

5.3.2　Web ADF 中的 ArcGIS Server API

Web ADF 中实现与 ArcGIS Server 交互的类包含在两个类库中，分别是 ESRI.ArcGIS.ADF.ArcGISServer.dll 与 ESRI.ArcGIS.ADF.Web.DataSouce.ArcGISServer.dll。后者包含了 ArcGIS Server 本地与远程数据源的公有 API 的实现类。ArcGIS Server 本地数据源直接连接 SOM 以及直接与服务器对象交互。ArcGIS Server 远程数据源连接的是 ArcGIS Server Web 服务端点。在 Web ADF 中，两者都使用 ArcGIS Server SOAP API 来实现。

许多基础类，例如 MapResourceBase、GISDataSourceBase，为这两种 ArcGIS Server 数据源定义了共享的成员，但是这两不同的数据源使用不同的实现类与代理类连接 ArcGIS Server 服务，如图 5.7 所示。此外，就如同 ADF 9.0/9.1 的方式，ArcGIS Server 本地数据源能与服务器上下文交互，并能通过客户端的 ArcObjects COM 代理类，创建服务器上的对象。

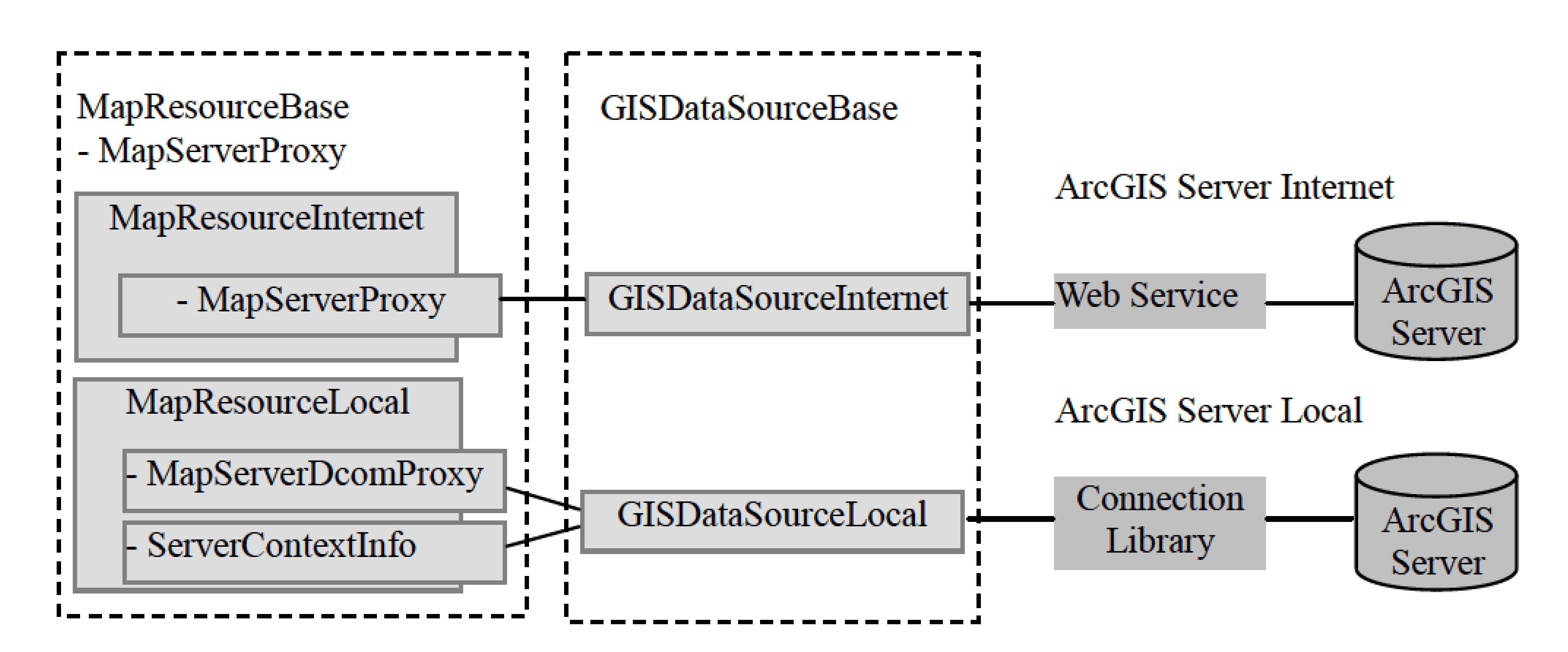


图 5.7　不同数据源使用不同的实现类与代理类连接

通过 ESRI ArcaIS ADF ARC GISServer.dll 程序集中的值对象与代理数，可使用 ArcaIS ServerSoAP API。两种数据源共用基于无状态的值对象，但使用不同的代理类与服务器交互。ArcGIS Server 本地数据源利用 DCOM 代理（例如 MapServerDcomProxy），直接与服务器对象通讯。而 ArcGIS Server 远程数据源基于 Web 服务代理（例如 MapServerProxy），与 Web 服务端口通讯。

值对象存储在客户端，依赖于客户端的环境。代理对象也存储在客户端，它负责与服务器端远程对象的通讯。根据需要，代理对象根据远程对象的要求序列化本地值对象，然后传递到服务端；对于服务端远程对象的结果反序列化，并在本地创建值对象，过程如图 5.8 所示。

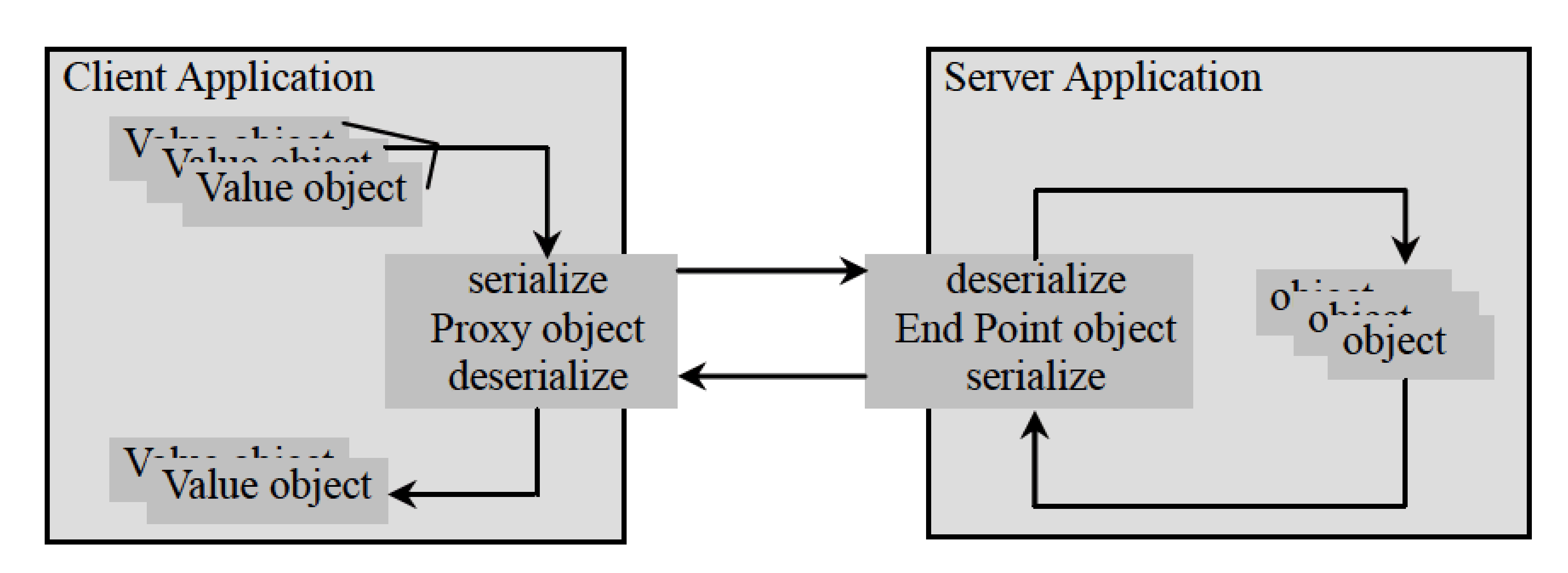


图 5.8 代理对象的序列化与反序列化

Web 服务工具使用 WSDL 文档，生成本地值对象和 Web 服务的代理对象。在 Visual Stuido 2005 以及.NET 开发包中的 wsdl.exe 就是这样一个工具。

每类 ArcGIS Server 服务都维护一个单独的 WSDL 文档，该文档描述了如何通过 SOAP 与服务交互。Web 服务代理只用于以无状态方式与 ArcGIS Server Web 服务交互。因此需要开发人员在客户端管理状态。在客户端，ArcGIS Server Web 服务的代理类负责 SOAP 的序列化与反序列化。在服务端，ArcGIS Server Web 服务管理服务器的上下文。对于 Web 服务，所有的通信都是无状态的，当处理 HTTP 请求时，创建服务器上下文，当响应发送给客户端时则销毁上下文。

如图 5.9 所示，尽管 ArcGIS Server 服务器对象可使用值对象，但是由于需要 COM 代理，因此通信要相对复杂。WSDL 没有规定 DCOM 代理直接与服务器对象交互，因此需要开发人员管理通信过程。如果服务器对象实现了 IFequestHandler 接口，使用 HandleStringRequest 方法来发送与获取 SOAP 请求与响应。也需要由开发人员负责初始化连接并序列化与反序列化 SOAP。如果使用了值对象，也需要负责它们的序列化与反序列化。不过 Web ADF 提供了一 DCOM 代理与一组相关的类帮助开发人员实现上述功能。

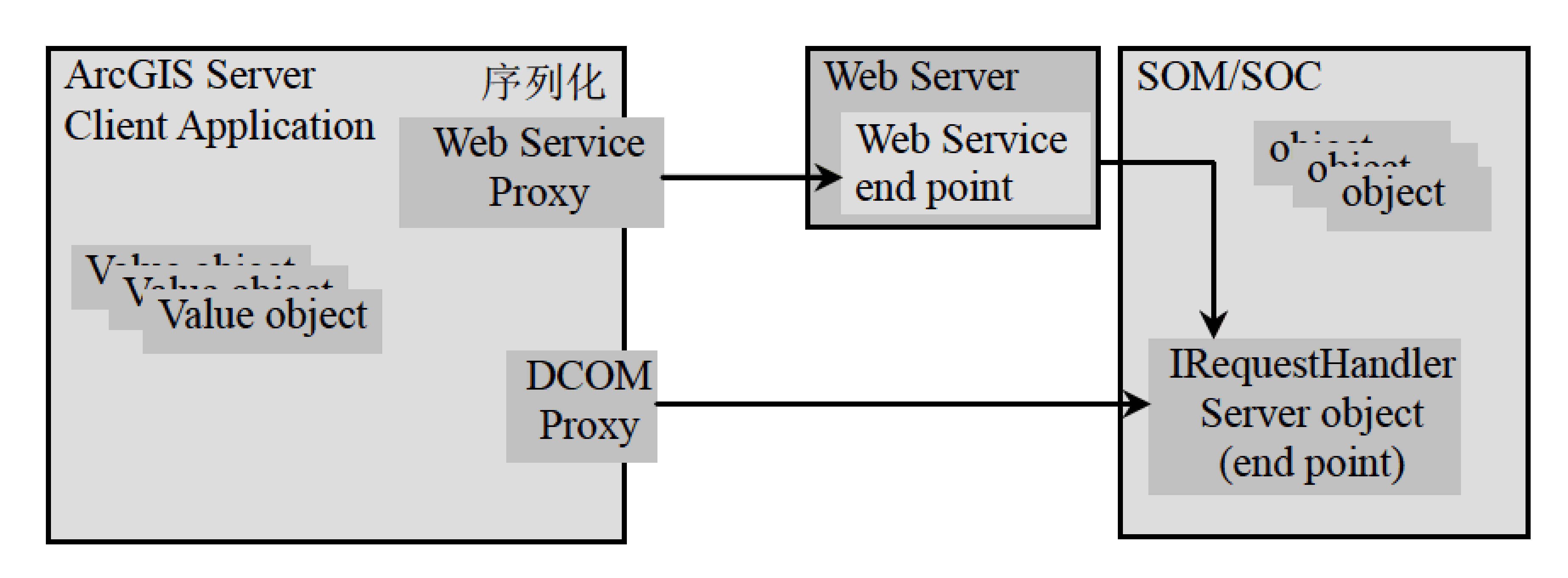


图 5.9 ArcGIS Server 代理类与数据源的连接

如图 5.10 所示，ArcGIS Server 本地数据源使用 DCOM 代理将值对象序列化为 SOAP，并通过 IRequestHandler 接口与服务器对象交互。也就是基于 DCOM 的 SOAP。例如，在 Web ADF 的地图控件中，如果是使用 ArcGIS Server 本地数据源，MapResourceLocal 实例使用 MapServerDcomProxy 类通过 SOAP 与服务器对象通信。而 ArcGIS Server 远程数据源使用 Web 服务代理将值对象序列化为 SOAP，并通过 HTTP 与 Web 服务端点交互。也就是基于 HTTP 的 SOAP。例如，使用 ArcGIS Server 远程数据源的地图控件，MapResouceInternet 实例使用 MapServerProxy 类通过 SOAP 与 Web 服务通信。

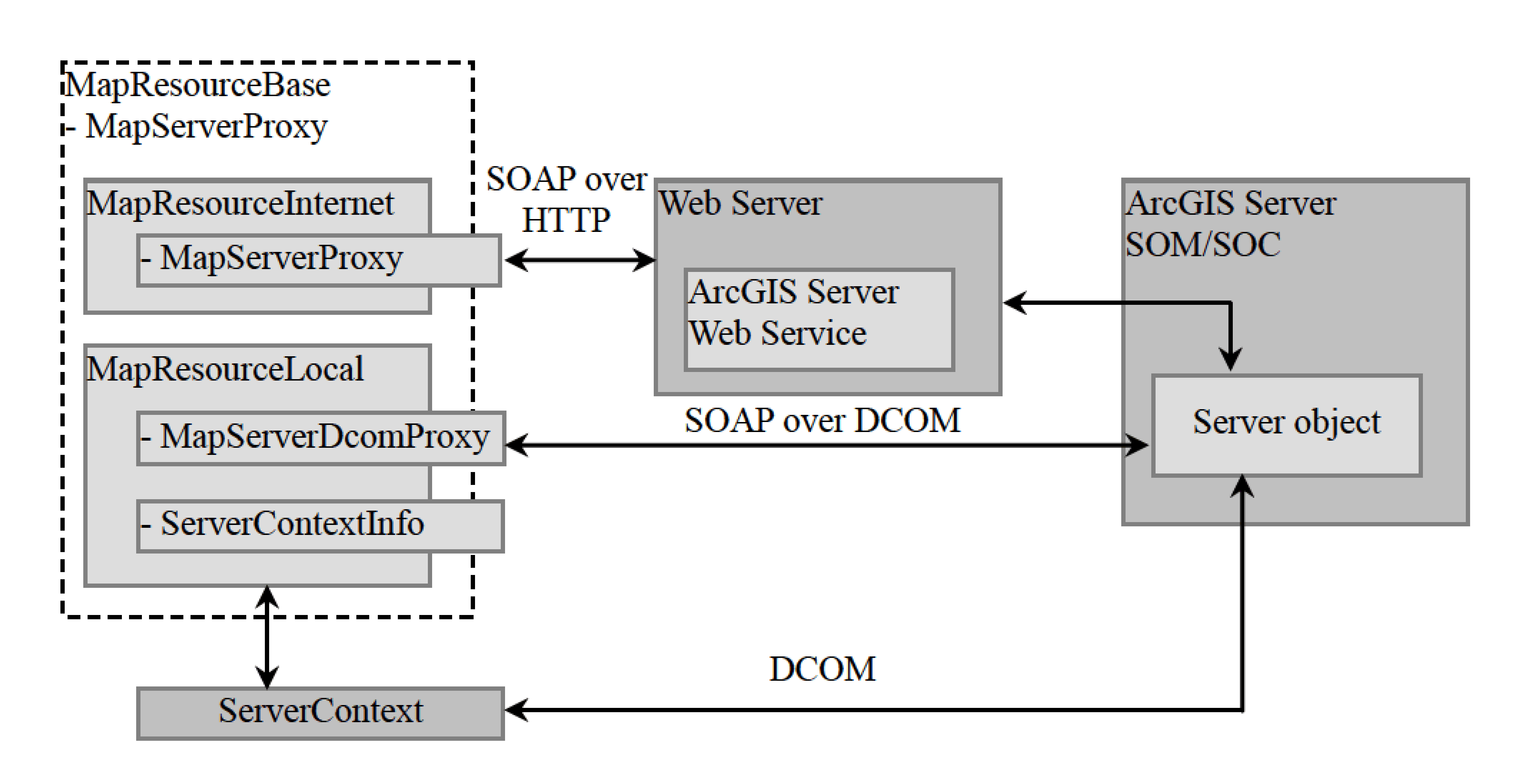


图 5.10　Web ADF 中值对象与代理类的使用

每个地图控件创建一个 MapFunctionality 对象与客户端的值对象 MapDescription 交互。MapFunctionality 使用该 MapDescription 与 ArcGIS Server 地图服务器对象的绘图功能打交道，例如生成一张新地图图片。MapFunctionality 对象的 MapDescription 属性存储了地图控件的状态，而 MapResourceLocal 对象的 MapDescription 属性存储了资源的状态，该值可以被使用该资源的任何功能共享。但默认时，每个 MapFunctionality 使用自己的 MapDescription 来维护状态。不管是哪种情况，MapResourceLocal 对象使用 MapServerDcomProxy 类创建服务器上下文，并管理与地图服务器对象的无状态的交互。MapResourceLocal 同时也提供了访问服务器上的细粒度的 ArcObjects。可通过 MapServer 属性服务 MapServer 对象，通过 ServerContextInfo 类可访问服务器上下文。

5.3.3　管理服务器上下文

当使用 ArcGIS Server 本地数据源时，可通过服务器上下文直接与服务器对象交互。服务可以发布为池化也可为非池化。当在地图资源管理器中，将 ArcGIS Server 服务器对象作为资源使用时，地图资源管理器创建与管理服务器上下文。对于池化的服务器对象，并不需要开发人员编写代码释放服务器上下文，而是由 Web ADF 来释放。但是对于非池化的服务器对象，需要通过代码释放。在服务器上下文释放之前，表示服务器对象实例仍然在使用，其他客户不能使用。

下面我们通过一个实例演示如何获取服务器上下文的。在 Visual Studio 2005 中，利用 File 菜单的 New Web Site 命令，创建一个新的 Web 站点，命名为 ContextManager。为了演示如何使用 ArcGIS Server ArcObjects API，本实例不使用任何 Web ADF 控件。这就要求我们手工加入如下 ArcGIS 引用（通过工程的右键菜单的 Add ArcGIS Reference 命令）：ESRI.ArcGIS.Server、ESRI.ArcGIS.ADF.Connection、ESRI.ArcGIS.Carto、ESRI.ArcGIS.ADF、ESRI.ArcGIS.Geodatabase、ESRI.ArcGIS.Display 与 ESRI.ArcGIS.Geometry。

在 Default.aspx 中加入一个 Image 控件，将其 ID 设置为 imgOutput。

切换到 Default.aspx.cs 文件中，在头部加入如下命名空间的引用：

```
Using ESRI.ArcGIS.ADF
using ESRI.ArcGIS.Server;
using ESRI.ArcGIS.Carto;
using ESRI.ArcGIS.ADF.Connection.AGS;
using ESRI.ArcGIS.Geodatabase;
using ESRI.ArcGIS.Display;
```

首先加入如下代码，以获取服务器上下文对象：

```
private IServerContext getServerContext() {
    string servername = "localhost";
    string mapserverobject = "USAMap";
    IServerObjectManager serverManager;

    if (Session["SOM"] == null) {
        // *** 使用 Web ADF 公有 API
        Identity identity = new Identity("Administrator", "1", null);
        AGSServerConnection gisconnection =
           new AGSServerConnection(servername,
            identity);
        gisconnection.Connect();

        serverManager = gisconnection.ServerObjectManager;
        Session.Add("SOM", serverManager);
    }
    else {
        serverManager = Session["SOM"] as IServerObjectManager;
    }

    IServerContext mapContext =
       serverManager.CreateServerContext(mapserverobject, "MapServer");
    return mapContext;
}
```

要得到服务器上下文对象，就必须先建立与服务器的连接。可以使用 AGSServerConnection 类来建立连接，在构造该类的对象时，第一个参数是服务器的计算机名或 IP 地址，第二个参数是一个 Identity 对象，用于身份验证。Identity 类构造函数的第一个参数是用户名，第二个参数是口令，第三个参数是域名，一般可以不提供。

在构造好连接对象后，调用其 Connect 方法进行连接。连接成功后通过连接对象的 ServerObjectManager 属性找到 GIS 服务器上的 SOM。然后通过 SOM 创建一个服务器上下文对象。

服务器上下文本质上是一个 GIS 服务器上的进程，它也是服务器端编程的起点。因此，我们是通过 CreateServerContext 命令在服务器端上创建，而不使用 new 关键字在本机上创建。

加入页面 PreRender 事件处理代码，如下所示：

```
protected void Page PreRender(object sender, EventArgs e) {
    if (Page.IsPostBack)
        return;

    // 得到服务器上下文对象
```

```
        IServerContext mapContext = getServerContext();
        IMapServer mapServer = mapContext.ServerObject as IMapServer;

        IMapServerInfo mapInfo =
          mapServer.GetServerInfo(mapServer.DefaultMapName);
        IMapDescription mapDesc = mapInfo.DefaultMapDescription;

        CreateMapImage(mapContext, mapDesc);

        mapContext.ReleaseContext();
    }
```

在上面的代码中，首先是调用 getServerContext 方法，得到服务器上下文对象，然后从上下文对象找到服务器对象，从服务器对象中得到服务器地图信息对象，该对象的 DefaultMapDescription 属性维护着地图描述信息。然后调用自定义方法 CreateMapImage 得到地图，最后调用 ReleaseContext 方法释放上下文对象。

定义 CreateMapImage 方法的代码如下：

```
    private void CreateMapImage(IServerContext mapContext, IMapDescription md)
    {
        IImageType imageType;
        IImageDescription imageDesc;
        IImageDisplay imageDisp;

        imageType = mapContext.CreateObject("esriCarto.ImageType")
                    as IImageType;
        imageDesc = mapContext.CreateObject("esriCarto.ImageDescription")
                    as ImageDescription;
        imageDisp = mapContext.CreateObject("esriCarto.ImageDisplay")
                    as ImageDisplay;

        imageType.Format = esriImageFormat.esriImageJPG;
        imageType.ReturnType = esriImageReturnType.esriImageReturnURL;

        imageDisp.Height = (int)imgOutput.Height.Value;
        imageDisp.Width = (int)imgOutput.Width.Value;
        imageDisp.DeviceResolution = 96;

        imageDesc.Display = imageDisp;
        imageDesc.Type = imageType;

        IMapServer mapServer = (IMapServer)mapContext.ServerObject;

        IImageResult mapImage = mapServer.ExportMapImage(md, imageDesc);
        imgOutput.ImageUrl = mapImage.URL;
        imgOutput.Visible = true;
    }
```

在上述代码中，通过上下文对象的 CreateObject 方法创建了几个与绘图相关的对象。这里要特别注意的是，由于我们是要在远程的 GIS 服务器上创建对象，而不是在本应用程序的进程中创建

对象，所以不能使用 new 关键字，而必须使用上下文对象的 CreateObject 方法来创建。

在创建好绘图需要的相关对象后，设置恰当的属性值，最后调用服务器对象的 ExportMapImage 方法输出地图图片。

编译并运行程序，程序运行效果如图 5.11 所示。

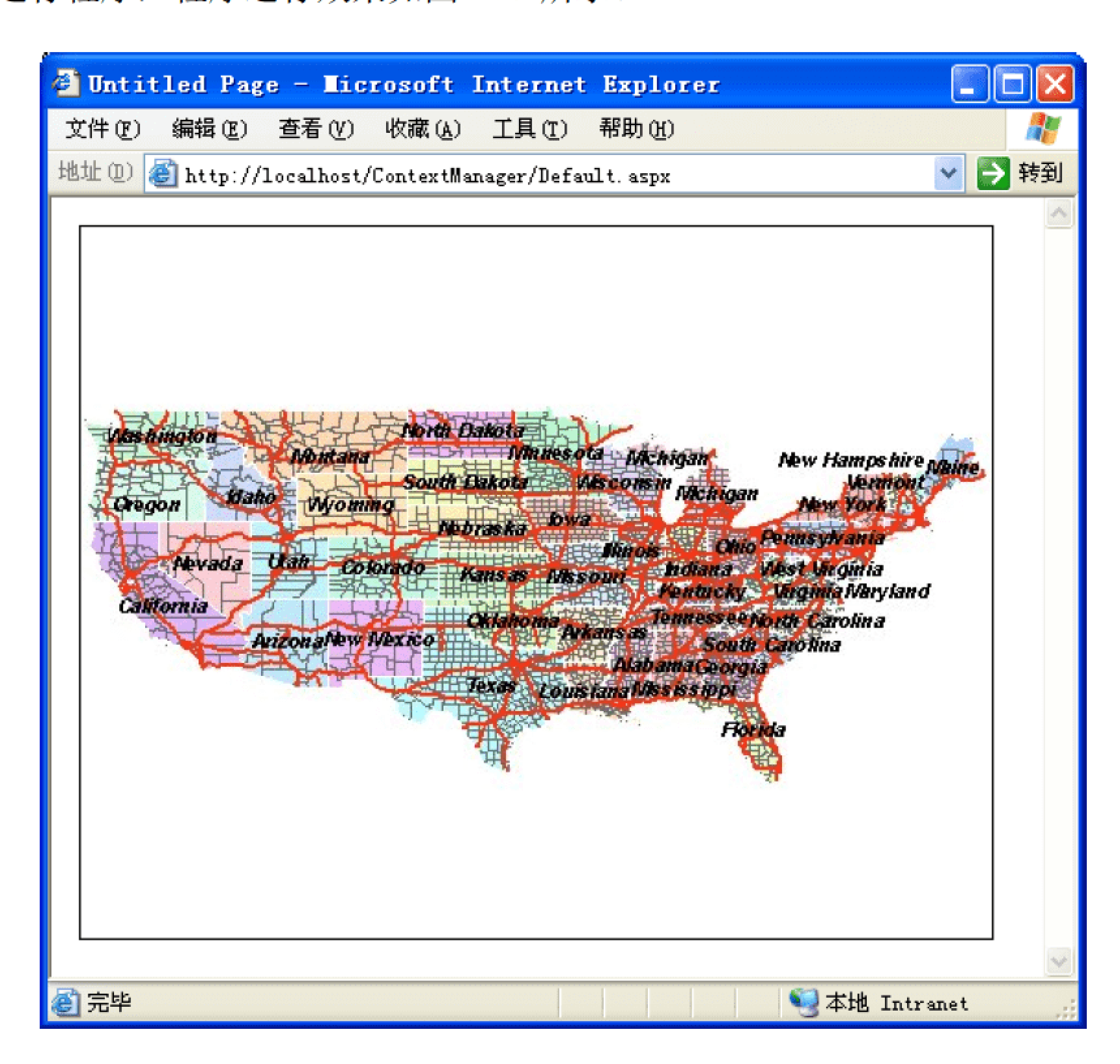


图 5.11 利用服务器上下文得到地图

下面再演示如何通过程序动态地增加图层数据。在 Default.aspx.cs 页面文件中增加如下方法，实现增加图层：

```
private void AddTheLayers(IServerContext in mapcontext)
{
    IFeatureLayer layer = (IFeatureLayer)in mapcontext.CreateObject(
                "esriCarto.FeatureLayer");

    IWorkspaceFactory factory =
       (IWorkspaceFactory)in mapcontext.CreateObject(
       "esriDataSourcesFile.ShapefileWorkspaceFactory");

    string filepath = @"C:\ProgramFiles\ArcGIS\DeveloperKit\
                      SamplesNET\Server\data\NorthAmerica";
    string filename = "canada.shp";
    IWorkspace ws = factory.OpenFromFile(filepath, 0);
    IFeatureWorkspace fws = (IFeatureWorkspace)ws;
    layer.FeatureClass = fws.OpenFeatureClass(filename);
    layer.Name = "canada";
```

```
        IGeoFeatureLayer iglayer = (IGeoFeatureLayer)layer;
        IFeatureRenderer renderer = iglayer.Renderer;

        ISimpleRenderer isr = (ISimpleRenderer)renderer;
        IRgbColor irgbc =
           (IRgbColor)in mapcontext.CreateObject("esriDisplay.RgbColor");
        irgbc.Red = 0;
        irgbc.Green = 0;
        irgbc.Blue = 210;

        ESRI.ArcGIS.Geometry.esriGeometryType featype =
          layer.FeatureClass.ShapeType;
        if (featype == ESRI.ArcGIS.Geometry.esriGeometryType.esriGeometryPoint)
        {
            ISimpleMarkerSymbol ifs = (ISimpleMarkerSymbol)isr.Symbol;
            ifs.Color = (IColor)irgbc;
        }
        else if (
        featype == ESRI.ArcGIS.Geometry.esriGeometryType.esriGeometryPolyline){
            ISimpleLineSymbol ifs = (ISimpleLineSymbol)isr.Symbol;
            ifs.Color = (IColor)irgbc;
        }
        else if (
        featype == ESRI.ArcGIS.Geometry.esriGeometryType.esriGeometryPolygon){
            ISimpleFillSymbol ifs = (ISimpleFillSymbol)isr.Symbol;
            ifs.Color = (IColor)irgbc;
        }
        else {
            throw new Exception("图层类型不确定！");
        }

        IMapServer ms = (IMapServer)in_mapcontext.ServerObject;

        IMapServerObjects2 mso = (IMapServerObjects2)ms;
        string mapname = ms.DefaultMapName;
        IMap map = mso.get Map(mapname);

        map.AddLayer(layer);
        mso.RefreshServerObjects();
    }
```

上述代码中，首先利用服务器上下文对象，创建要素图层对象与工作空间工厂对象，然后利用此工厂对象打开一文件夹，作为工作空间对象。利用此工作空间对象的 OpenFeatureClass 方法，将文件夹中指定文件名的要素图层打开。接着指定着色器对象。最后将图层增加到地图中，完成动态增加图层。

然后在 Default.aspx 页面中增加一个按钮控件，ID 设置为 AddLayer。双击该控件，生成控件的 OnClick 事件处理方法，在其中加入如下代码：

```
protected void AddLayer_Click(object sender, EventArgs e) {
    // 得到服务器上下文对象
    IServerContext mapContext = this.getServerContext();

    // 增加图层
    AddTheLayers(mapContext);

    // 从上下文对象中得到服务器对象
    IMapServer ms = (IMapServer)mapContext.ServerObject;
    IMapServerInfo mapInfo = ms.GetServerInfo(ms.DefaultMapName);
    IMapDescription NEWmapDesc = mapInfo.DefaultMapDescription;

    // 输出地图
    CreateMapImage(mapContext, NEWmapDesc);

    // 释放服务器上下文对象
    mapContext.ReleaseContext();
}
```

代码很简单，首先得到服务器上下文对象，然后加入图层，接着输出一张地图，最后释放对象。

编译并运行程序，通过选择“增加图层”按钮，即可动态增加加拿大图层数据，所得结果如图 5.12 所示。

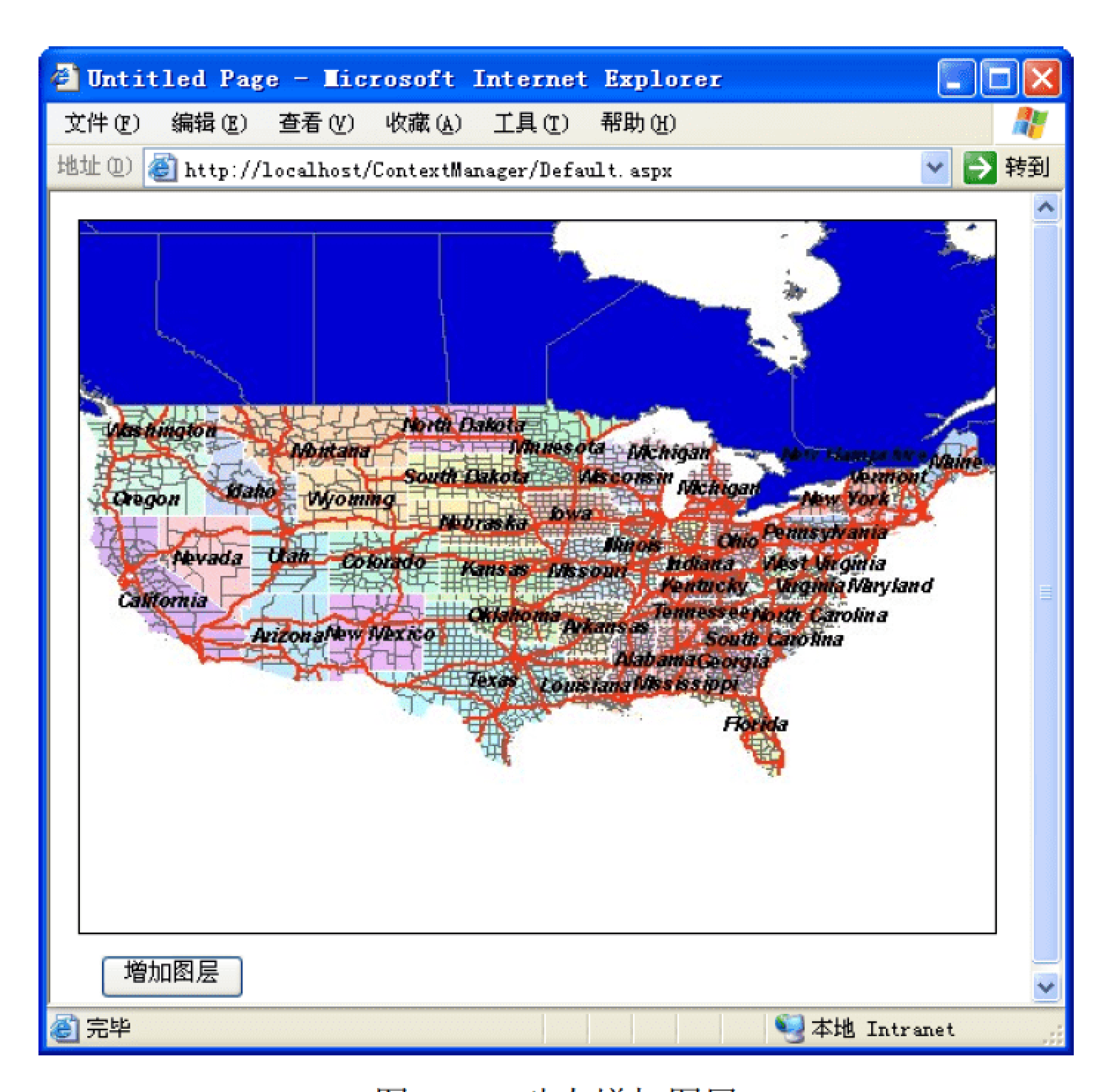


图 5.12 动态增加图层

服务器上下文对象除了用我们上面演示的 CreateObject 方法来创建对象之外，可用 SetObject 和 GetObject 方法存储与获取对象的引用。每个服务器上下文对象都包含一个字典(存储名称和值)，该字典是一个用来在上下文对象内保存在上下文对象内创建的对象最方便的地方。只要上下文对象存在，其字典中存储的对象就存在，当释放上下文对象以后其字典也就不存在了。可以在程序的不

同地方通过该字典来共享数据。例如：

```
IPointCollection pPointCollection =
    pServerContext.CreateObject("esriGeometry.Polygon") as IPointCollection;

// 先在字典中存储对象 pPointCollection
pServerContext.SetObject ("myPoly", pPointCollection);
//从字典中获取对象 pPointCollection
IPolygon pPoly = pServerContext.GetObject("myPoly");
```

要从字典中删除对象可以使用 Remove 和 RemoveAll 方法。

SaveObject 和 LoadObject 方法用来在服务器上下文对象之间共享数据，SaveObject 把某个上下文对象中创建的对象序列化为字符串，LoadObject 用来在另一个上下文对象中把某个字符串反序列化为对象。例如：

```
IServerContext pServerContext =
    pSOM.CreateServerContext("RedlandsMap", "MapServer");
IServerContext pServerContext2 =
    pSOM.CreateServerContext("RedlandsGeocode", "GeocodeServer");
IGeocodeServer pGCServer = pServerContext2.ServerObject;
IPropertySet pPropertySet =
    pServerContext2.CreateObject("esriSystem.PropertySet");
pPropertySet.SetProperty("Street", "");
IPropertySet pResults = pGCServer.GeocodeAddress(pPropertySet, Nothing);
IPoint pPoint = pResults.GetProperty("Shape");

// 把 pServerContext2 中的 pPoint 对象序列化
string sPoint = pServerContext2.SaveObject(pPoint);
//在 pServerContext 中把 sPoint 字符串反序列化
IPoint pPointCopy = pServerContext.LoadObject(sPoint);
```

总之，SetObject 和 GetObject 是在同一个上下文对象中共享数据。SaveObject 和 LoadObject 方法是在多个上下文对象之间共享数据。

5.4　操作资源与功能对象

资源的初始化与管理都由资源管理器来完成。Web ADF 提供了三个资源管理器，分别是地图资源管理器、地理编码资源管理器与空间处理资源管理器。每个资源管理器包含一个资源集合。

5.4.1　增加与删除资源

可以在程序设计期间通过 Visual Studio 进行，也可以在程序运行期间通过代码，加入资源。第 4 章已经介绍了如何在设计期间加入资源。

要通过代码加入资源，首先需要创建资源项，然后设置起属性，最后加入到资源管理器中。下面通过一个实例来介绍如何动态管理资源管理器中的资源项。

在 Visual Studio 2005 中，利用 File 菜单的 New Web Site 命令，创建一个新的 Web 站点，命名为 DynamicAddResource。

从工具箱的 ArcGIS Web Controls 选项卡中，在 Default.aspx 页面中增加一个地图资源管理器控件、一个地图控件以及一个 Toc 控件。在地图资源控件中加入 NorthAmericaMap 资源。并通过工程右键菜单的 Add ArcGIS Identity 命令加入身份验证信息。

再从工具箱的 Standard 选项卡中，加入一 CheckboxList 控件，通过其 Items 属性，在其中加入两项，分别为将其 Text 属性设置为“ArcGIS Server 本地资源”与“ArcGIS Server 远程资源”。

在 Default.aspx.cs 文件的头部加入如下命名空间的引用：

```
using ESRI.ArcGIS.ADF.Web.UI.WebControls;
using ESRI.ArcGIS.ADF;
using ESRI.ArcGIS.ADF.Web;
using System.IO;
```

为使用回调机制，需要_Default 类实现 ICallbackEventHandler 接口。

在_Default 类中加入如下字段：

```
public string addRemoveResource;
private string resourceName = string.Empty; // 选中的资源名称
private string callbackResponse = "";  // 回调返回的字符串
const string AGSLocalName = "LocalMapResource";
const string AGSInternetName = "InternetMapResource";
```

在 Page_OnLoad 方法中加入如下代码：

```
protected void Page Load(object sender, EventArgs e) {
    foreach (ListItem li in CheckBoxList1.Items) {
        li.Attributes.Add("onclick", "ChangeCheckContext(this)");
    }

    addRemoveResource = ClientScript.GetCallbackEventReference(this,
        "message", "processCallbackResult", "context", "postBackError", true);
}
```

通过上面的代码，指定了复选框的 onclick 事件的处理函数为 ChangeCheckContext。

切换到 Default.aspx 中，在其<head>与</head>之间加入 ChangeCheckContext 函数的代码，代码如下：

```
<script language="javascript" type="text/javascript">
function ChangeCheckContext(checkbox) {
    var val = checkbox.nextSibling.innerText;
    var ischecked = checkbox.checked;
    context = "cb";
    var message = "changeresource,";

    message += val + ',' + ischecked;

    <%= addRemoveResource%>
 }
</script>
```

在_Default 类中加入如下实现 ICallbackEventHandler 接口的代码：

```
string ICallbackEventHandler.GetCallbackResult() {
    return callbackResponse;
}

void ICallbackEventHandler.RaiseCallbackEvent(string eventArgs) {
    char[] charToParse = { ',' };
    string[] messages = eventArgs.Split(charToParse);
    if (eventArgs.Contains("changeresource")) {
        string dataSourceType = messages[1];
        bool isChecked = Boolean.Parse(messages[2]);
        switch (dataSourceType) {
            case "ArcGIS Server 本地资源":
                resourceName = AGSLocalName;
                break;
            case "ArcGIS Server 远程资源":
                resourceName = AGSInternetName;
                break;
            default:
                break;
        }

        MapResourceChange(isChecked);
    }
}
```

在进行回调处理时，代码主要集中在 RaiseCallbackEvent 方法。在该方法中先判断用户选择的是哪个复选框，并判断是选择还是去除选择，最后调用 MapResourceChange 方法实现资源的动态增加与删除。该方法的代码如下：

```
private void MapResourceChange(bool isChecked)
{
    GISResourceItemCollection<MapResourceItem> mapResourceItemCollection =
        MapResourceManager1.ResourceItems;
    MapResourceItem mapResourceItem = null;

    // 如果选中，则增加资源；否则，删除资源
    if (!isChecked) {
        mapResourceItem = mapResourceItemCollection.Find(resourceName);
        mapResourceItemCollection.Remove(mapResourceItem);
    }
    else {
        mapResourceItem = new MapResourceItem();

        // 地图资源项由一定义与显示设置组成
        // 定义规定了数据源与资源参数
        // 显示设置规定了地图图片的属性
        GISResourceItemDefinition definition =
            new GISResourceItemDefinition();
```

```
            switch (resourceName) {
                case (AGSLocalName):
                    mapResourceItem.Name = AGSLocalName;
                    definition.DataSourceDefinition = "localhost";
                    definition.DataSourceType = "ArcGIS Server Local";
                    definition.ResourceDefinition = "Layers@USAMap";
                    break;
                case (AGSInternetName):
                    mapResourceItem.Name = AGSInternetName;
                    definition.DataSourceDefinition =
                          "http://localhost/arcgis/services/";
                    definition.DataSourceType = "ArcGIS Server Internet";
                    Identity id = new Identity("Administrator", "1", "");
                    definition.Identity = id.ToString();
                    definition.ResourceDefinition = "(default)@USAMap";
                    break;
            }

            definition.DataSourceShared = true;
            mapResourceItem.Parent = MapResourceManager1;
            mapResourceItem.Definition = definition;

            DisplaySettings displaySettings = new DisplaySettings();
            displaySettings.Transparency = 50.0F;
            displaySettings.Visible = true;
            displaySettings.ImageDescriptor.TransparentColor =
               System.Drawing.Color.White;
            displaySettings.ImageDescriptor.TransparentBackground = true;
            mapResourceItem.DisplaySettings = displaySettings;

            // 将新资源项插入到开始位置，即最上面
            MapResourceManager1.ResourceItems.Insert(0, mapResourceItem);
            // 创建该地图资源
            ESRI.ArcGIS.ADF.Web.DataSources.IMapResource mapResource =
                MapResourceManager1.CreateResource(mapResourceItem);


            // 依据融合位置不同，刷新地图或资源
            if (Map1.ImageBlendingMode == ImageBlendingMode.WebTier) {
                Map1.Refresh();
            }
            else {
                Map1.RefreshResource(mapResourceItem.Name);
            }
        }


        // 刷新 Toc 控件，并将回调信息加入到地图控件的回调集合中
        Toc1.Refresh();
        CallbackResult tocCallbackResult = RefreshControlHtml(Toc1);
```

```
        Map1.CallbackResults.Add(tocCallbackResult);

        // 将地图控件的回调集合转换为字符串
        callbackResponse = Map1.CallbackResults.ToString();
    }
```

在上述代码中，对于移除某一个资源，代码很简单，只需要从地图资源管理器对象的 ResourceItems 属性中删除即可。对于新增加资源，则新构造一个 MapResourceItem 对象，然后设置该对象的 Definition 与 DisplaySetting 属性，再将该对象插入到地图资源管理器对象的 ResourceItems 属性中。接着调用地图资源管理器对象的 CreateResource 方法创建新加入的地图资源。最后刷新地图与 Toc 控件，并返回客户端更新信息。

在上述方法中调用了 RefreshControlHtml 方法，用于将某控件的刷新转换为 HTML 语句，代码如下：

```
    private CallbackResult RefreshControlHtml(Control control)
    {
        StringWriter stringWriter = new StringWriter();
        HtmlTextWriter writer = new HtmlTextWriter(stringWriter);
        control.RenderControl(writer);
        string htmlContent = stringWriter.ToString();
        stringWriter.Close();
        return new CallbackResult(control, "content", htmlContent);
    }
```

编译并运行程序，通过复选框便可动态地加入或删除地图资源了，效果如图 5.13 所示。

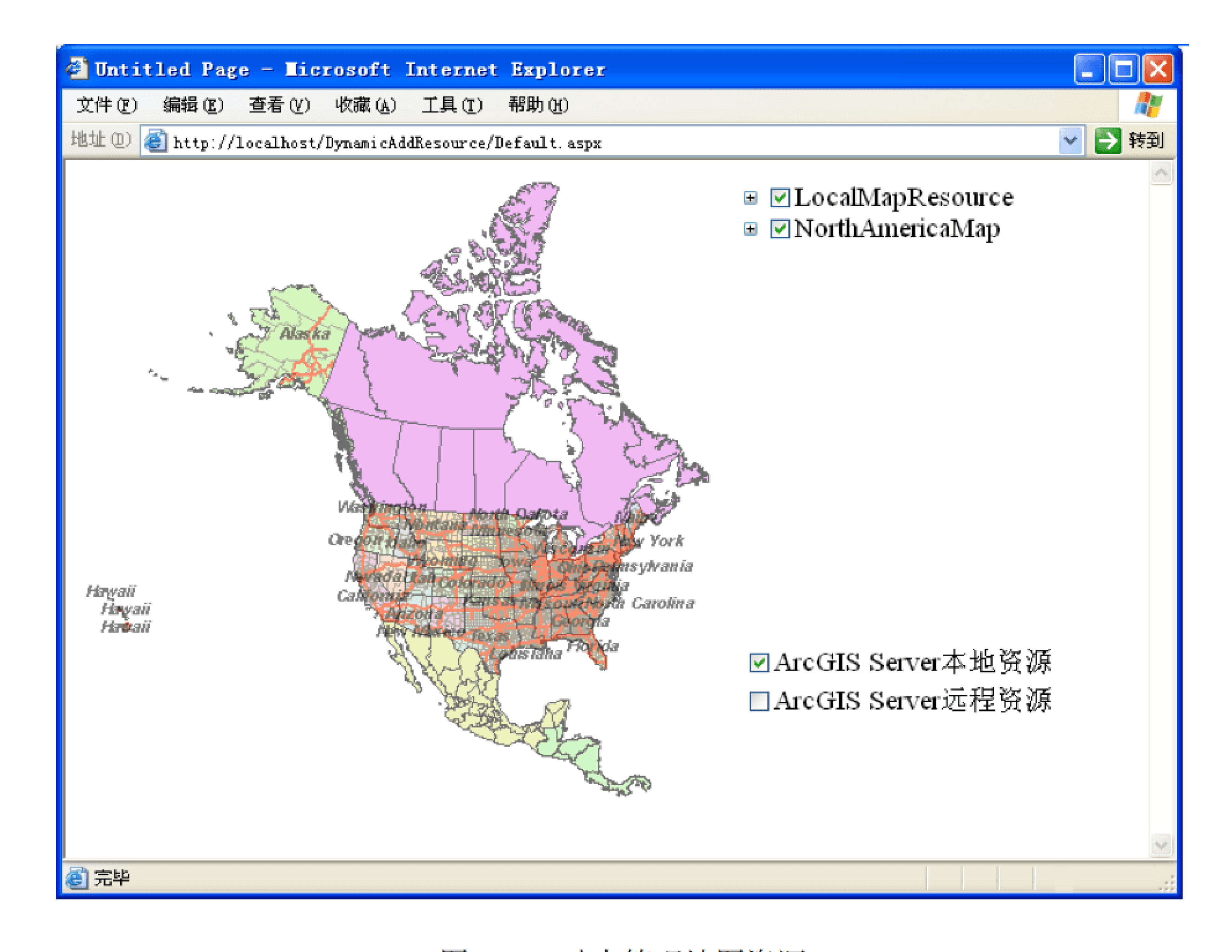


图 5.13　动态管理地图资源

5.4.2 管理地图资源

地图资源代表可访问地图数据，这些地图数据包括地图图片（动态的与静态的）、要素符号与着色器、以及要素的属性。IMapResource 接口中定义了抽象的属性存储显示设置与地图信息。

在 Web ADF 中，管理地图资源的是 MapResourceManager 控件，即前面经常用到的地图资源管理器。该资源可创建 IMapFunctionality、IQueryFunctionality、IMapTocFunctionality、ITileFunctionality 与 IScaleBarFunctionality 功能，分别代表绘图、查询、在 Toc 中显示、地图切片以及显示比例功能。

表 5-4 列出了 IMapResource 接口中主要的属性与方法。

表 5-4 IMapResource 接口中主要的属性与方法

属性或方法	说明
DisplaySettings	访问或修改地图显示属性，例如透明程序、背景颜色以及可见性
MapInformation	获取地图资源范围与空间参考的默认值
CreateFunctionality()	创建功能，例如查询功能
SupportsFunctionality()	判断该资源是否支持某类型的功能

下面的代码演示了通常情况下，如何不需要知道具体的数据源，便可访问地图资源，并改变地图资源的属性。代码首先从地图资源管理器中得到第一个资源的对象，然后将其可见属性设置为 true，并将该资源所生成的地图图片透明程度设置为 50%。

```
IMapResource mapresource = MapResourceManager1.GetResource(0);
mapresource.DisplaySettings.Visible = true;
mapresource.DisplaySettings.Transparency = 50.0F;
```

上述代码使用的是公有 API 中的 IMapResource 接口。此外，数据源专有的地图资源实现类可提供更多数据源特有的属性。要得到专有地图资源的引用，只需进行类型转换即可，例如下面的代码将公有 IMapResource 接口转换为 MapResourceLocal 类型，当然能转换的前提是该资源确实是 ArcGIS Server 本地数据资源：

```
IMapResource mapresource = MapResourceManager1.GetResource(0);
MapResourceLocal mr = mapResource as MapResourceLocal;
```

由于 ArcGIS Server 本地数据源与远程数据源都通过 ArcGIS Server SOAP API 与服务工作，它们一些属性与方法是一样的，因此 Web ADF 为这两类数据源提供了一个基类，即 MapResourceBase。通常，这两者在 Web ADF 中都使用值对象，使用代理类与 GIS 服务器通信。表 5-5 列出了它们的共同属性。

表 5-5 ArcGIS Server 本地资源与远程资源的共同属性

属性	说明
MapDescription	访问或修改 MapDescription 值对象。Web ADF 利用该对象管理状态
Service	地图资源使用的地图服务的名称
MapServerProxy	用于管理与 ArcGIS Server 服务之间的 SOAP 请求与响应

如果不需要使用 ArcGIS Server 本地或远程资源特有的功能，可以使用该基类即可。下面的代码演示了如何通过 MapResourceBase 类得到 ArcGIS Server 服务的名称：

```
    IMapResource mapresource = MapResourceManager1.GetResource(0);
    ESRI.ArcGIS.ADF.Web.DataSources.ArcGISServer.MapResourceBase
ags_mapresourcebase =
      (ESRI.ArcGIS.ADF.Web.DataSources.ArcGISServer.MapResourceBase)
mapresource;
    string ags_servicename = ags_mapresourcebase.Service;
```

ArcGIS Server 远程资源是通过 Web 服务的端点与 ArcGIS Server 服务工作的，对应的类为 MapResourceInternet。由于只能通过 ArcGIS Server SOAP API 来连接 Web 服务（图 5.14），因此 ArcGIS Server 远程资源无法访问服务器上下文。

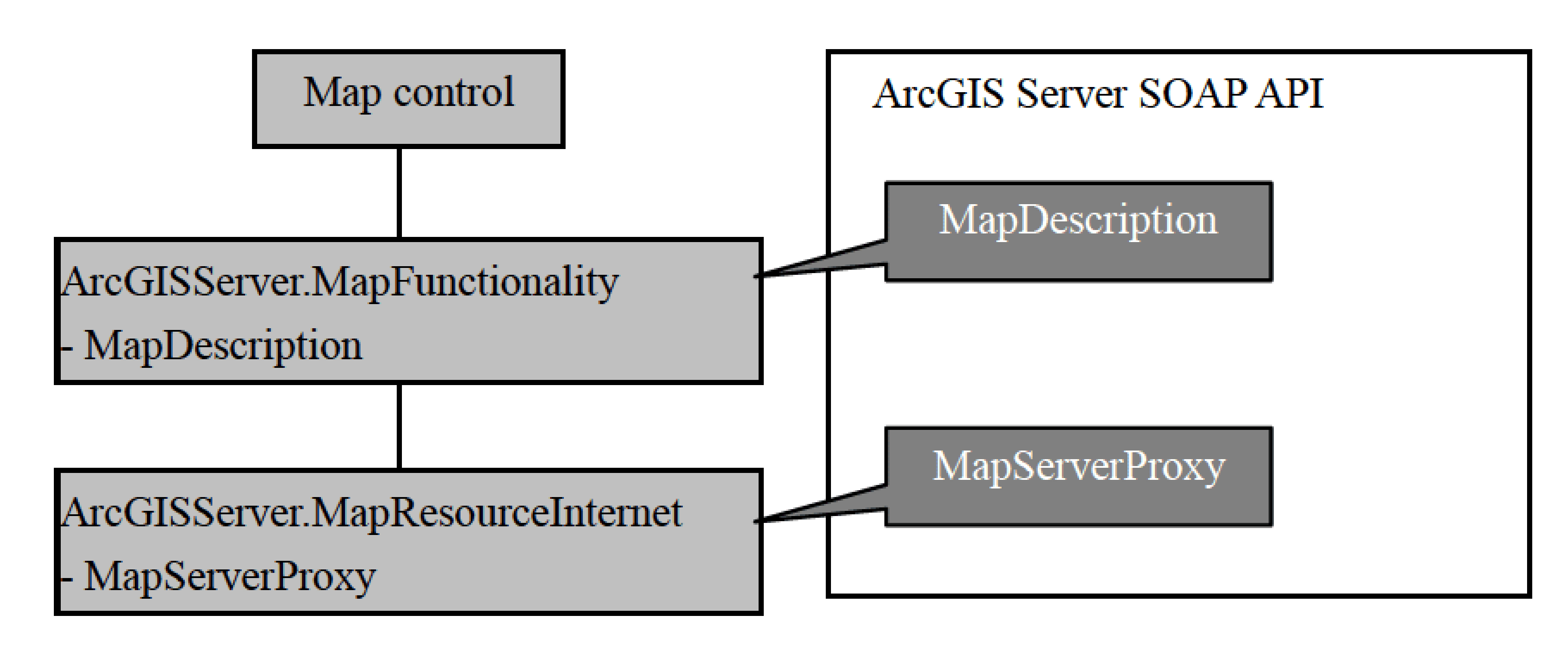


图 5.14　ArcGIS Server 远程资源与服务器只通过 SOAP API 连接

下面这段代码演示了如何通过远程资源得到 ArcGIS Server 服务的默认地图名称：

```
    IMapResource mapresource = MapResourceManager1.GetResource(0);
    ESRI.ArcGIS.ADF.Web.DataSources.ArcGISServer.MapResourceInternet
mapresource;
    mapresource = mapresource
         as ESRI.ArcGIS.ADF.Web.DataSources.ArcGISServer.MapResourceInternet ;
    ESRI.ArcGIS.ADF.ArcGISServer.MapServerProxy proxy =
         mapresource.MapServerProxy;
    string defaultmapname = proxy.GetDefaultMapName();
```

ArcGIS Server 本地资源可通过 ArcObjects API 中的 ServerContextInfo.ServerContext 属性来访问服务器上下文，并管理服务器对象，如图 5.15 所示。实现类为 MapResourceLocal。当将服务器对象作为 Web ADF 资源来使用时，并不需要创建与释放服务器上下文，而是由 Web ADF 的资源管理器控件来管理。

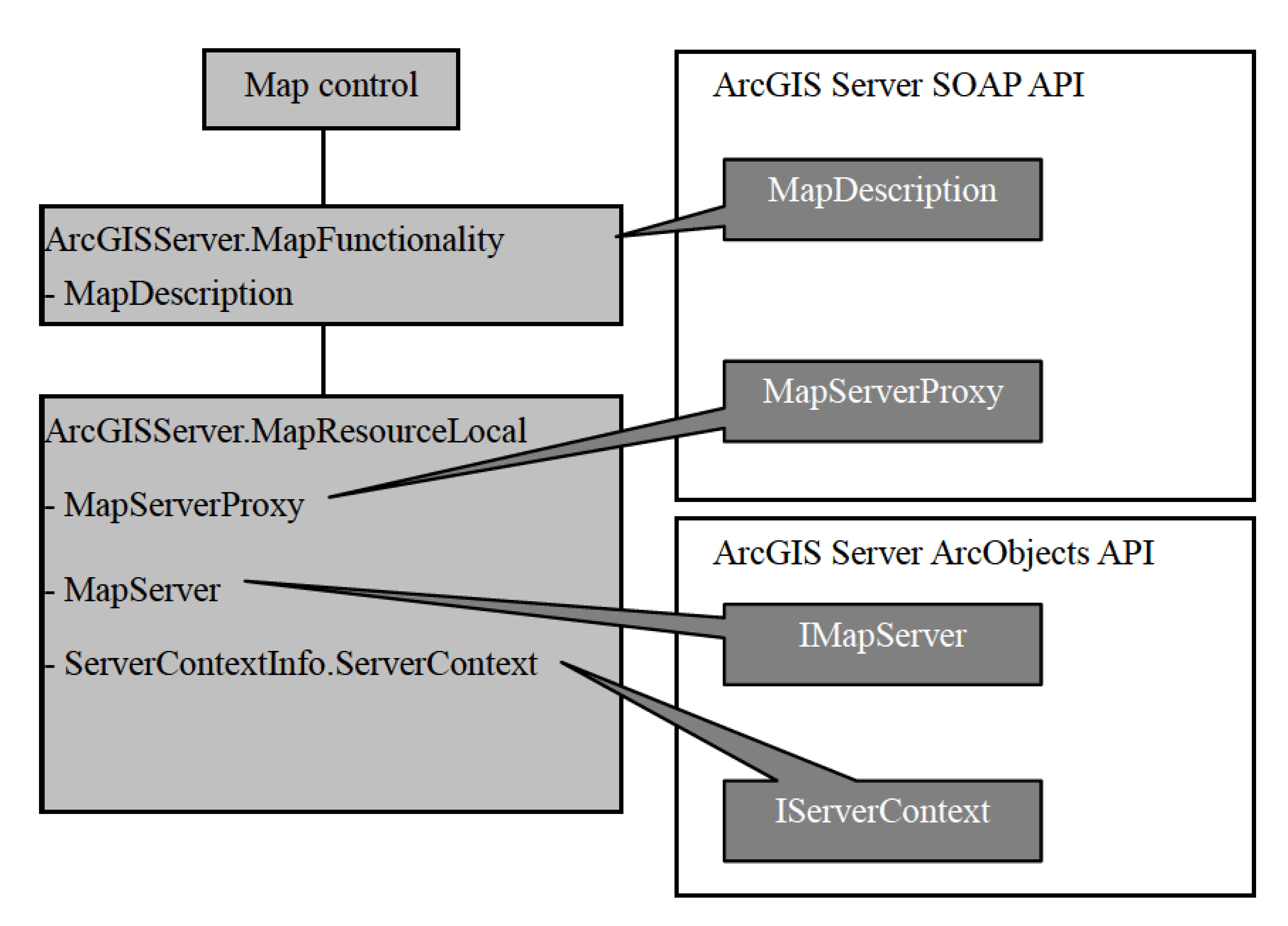


图 5.15　ArcGIS Server 本地资源可通过 ArcObjects API 连接

表 5-6 列出了 ArcGIS Server 本地资源专有的属性与方法。

表 5-6　ArcGIS Server 本地资源专有的属性与方法

属性与方法	说明
ServerContextInfo	服务器上下文信息对象，包括服务器上下文、服务器对象名称与类型
MapServer	用于访问 ArcObjects 的 IMapServer
MapServerProxy	用于访问 MapServerDcomProxy，管理通过 IRequestHandler 接口与服务器对象通信的 SOAP 请求与响应
ApplyMapDescriptionToServer()	当改变了 MapDescription 对象的属性后，调用该方法，改变 GIS 服务器中服务器对象实例的状态。
RefreshServerObjects()	改变 GIS 服务器中服务器对象实例的状态

下面通过一个实例来演示如何使用 MapResourceLocal。本实例要实现的功能完全与 5.3.3 节中 ContextManager 一样。只是在 ContextManager 中，没有通过 Web ADF，而是直接使用纯粹的 ArcObjects 类。

在 Visual Studio 2005 中，利用 File 菜单的 New Web Site 命令，创建一新的 Web 站点，命名为 AdfAddDynamicData。

从工具箱的 ArcGIS Web Controls 选项卡中，在 Default.aspx 页面中增加一地图资源管理器控件、一地图控件、一工具栏与 Toc 控件。在地图资源控件中加入 USAMap 资源。并通过工程右键菜单的 Add ArcGIS Identity 命令加入身份验证信息。用 4.1.3 节介绍的方法加入其他属性设置，使几个控件能联动。

再从工具箱的 HTML 选项卡中，加入两个 Input（Button）控件，分别设置其 id 属性为 AddLayer 与 RemoveLayer，value 属性为“增加图层”与“删除图层”，onclick 属性为 addlayer('addlayer')与 addlayer('removelayer') 。在 Default.aspx 代码的<head>与</head>之间加入如下代码，以响应用户选

择了上述两个按钮的操作：

```
<script language="javascript" type="text/javascript">
function addlayer(argument){
    var message = argument;
    var context = 'Map1';
    <%=addLayerCallback%>
}
</script>
```

在上面的 JavaScript 函数中，设置好 message 与 context 参数后，执行回调。

切换到 Default.aspx.cs 文件中，在头部加入如下命名空间：

```
using ESRI.ArcGIS.Server;
using ESRI.ArcGIS.Carto;
using ESRI.ArcGIS.Geodatabase;
using ESRI.ArcGIS.Geometry;
using ESRI.ArcGIS.Display;
using ESRI.ArcGIS.ADF.Web.DataSources.ArcGISServer;
```

为了应用回调机制，需要`_Default`类实现 ICallbackEventHandler 接口，因此将其声明代码行修改为如下代码：

```
public partial class _Default : System.Web.UI.Page, ICallbackEventHandler
```

在类中加入如下字段：

```
public string addLayerCallback;
private string callbackResponse;
MapResourceLocal localMapResource = null;
```

在 Page_Load 方法中加入如下代码，生成一个对客户端函数的引用：

```
protected void Page_Load(object sender, EventArgs e) {
    addLayerCallback = ClientScript.GetCallbackEventReference(this,
        "message", "processCallbackResult", "context", "postBackError", true);
}
```

接着需要完成的当然就是 ICallbackEventHandler 接口实现。该接口需要的两个方法的代码如下：

```
string ICallbackEventHandler.GetCallbackResult() {
    return callbackResponse;
}

void ICallbackEventHandler.RaiseCallbackEvent(string eventArgument) {
    if (eventArgument.Contains("addlayer")) {
        AddLayer();
    }
    else if (eventArgument.Contains("removelayer")) {
        Session["layeradded"] = false;
        RemoveLayer();
    }
}
```

主要工作是在 RaiseCallbackEvent 方法中完成的，该方法通过判断传入的参数是 addlayer 还是 removelayer，分别执行增加图层方法 AddLayer 与删除图层方法 RemoveLayer。

增加图层 AddLayer 方法的代码如下：

```
private void AddLayer() {
    MapFunctionality agsMapFunctionality = Map1.GetFunctionality(0)
                                as MapFunctionality;
    localMapResource = (MapResourceLocal)agsMapFunctionality.Resource;

    // 使用 ArcObjects 增加动态图层
    IServerContext mapContext =
       localMapResource.ServerContextInfo.ServerContext;

    IWorkspaceFactory workspaceFactory =
       (IWorkspaceFactory)mapContext.CreateObject(
       "esriDataSourcesFile.ShapefileWorkspaceFactory”);
    IFeatureWorkspace featureWorkspace =
       workspaceFactory.OpenFromFile(@"C:\ProgramFiles\ArcGIS\DeveloperKit\
                             SamplesNET\Server\data\NorthAmerica", 0)
                             as IFeatureWorkspace;
    IFeatureLayer featureLayer =
       (IFeatureLayer)mapContext.CreateObject("esriCarto.FeatureLayer”);
    featureLayer.FeatureClass =
       featureWorkspace.OpenFeatureClass("canada.shp");
    featureLayer.Name = "Canada";

    // 应用着色器
    ApplyRenderer(featureLayer, mapContext);

    IMapServerObjects mapServerObjects = localMapResource.MapServer
                              as IMapServerObjects;
    IMap map = mapServerObjects.get Map(localMapResource.DataFrame);
    map.AddLayer(featureLayer);
    map.MoveLayer(featureLayer, map.LayerCount - 1);

    // 刷新服务器对象，响应添加图层
    localMapResource.RefreshServerObjects();

    // 刷新地图与 Toc
    Toc1.Refresh();
    Map1.CallbackResults.CopyFrom(Toc1.CallbackResults);

    Map1.Refresh();
    callbackResponse = Map1.CallbackResults.ToString();
}
```

在上面的代码中，通过绘图功能得到 MapResourceLocal 对象，通过该对象的 ServerContextInfo 属性的 ServerContext 属性得到服务器上下文对象。然后利用该对象的 CreateObject 方法创建工作空间工厂对象与要素图层对象。在设置好图层的着色器以后，在地图中加入图层，接着调用本地资

源对象的 RefreshServerObjects 方法刷新服务器对象，最后刷新地图与 Toc。

其中 ApplyRenderer 方法的代码如下，用于为图层指定着色器：

```
// 设置动态要素图层的着色器对象
private void ApplyRenderer(IFeatureLayer featureLayer,
IServerContext mapContext) {
    IGeoFeatureLayer geoLayer = (IGeoFeatureLayer)featureLayer;

    ISimpleRenderer simpleRenderer = (ISimpleRenderer)
        mapContext.CreateObject("esriCarto.SimpleRenderer");
    IRgbColor rgbColor =
        (IRgbColor)mapContext.CreateObject("esriDisplay.RgbColor");
    rgbColor.Red = 0;
    rgbColor.Green = 0;
    rgbColor.Blue = 210;

    esriGeometryType geometryType = featureLayer.FeatureClass.ShapeType;
    if (geometryType == esriGeometryType.esriGeometryPoint)
    {
        ISimpleMarkerSymbol symbol = (ISimpleMarkerSymbol)
            mapContext.CreateObject("esriDisplay.SimpleMarkerSymbol");
        symbol.Color = (IColor)rgbColor;
        simpleRenderer.Symbol = (ISymbol)symbol;
    }
    else if (geometryType == esriGeometryType.esriGeometryPolyline) {
        ISimpleLineSymbol symbol = (ISimpleLineSymbol)
            mapContext.CreateObject("esriDisplay.SimpleLineSymbol");
        symbol.Color = (IColor)rgbColor;
        simpleRenderer.Symbol = (ISymbol)symbol;
    }
    else if (geometryType == esriGeometryType.esriGeometryPolygon) {
        ISimpleFillSymbol symbol = (ISimpleFillSymbol)
            mapContext.CreateObject("esriDisplay.SimpleFillSymbol");
        symbol.Color = (IColor)rgbColor;
        simpleRenderer.Symbol = (ISymbol)symbol;
    }
    else {
        throw new Exception("不能确定图层类型");
    }

    geoLayer.Renderer = (IFeatureRenderer)simpleRenderer;
}
```

删除图层 RemoveLayer 方法的代码如下：

```
private void RemoveLayer() {
    MapFunctionality agsMapFunctionality = Map1.GetFunctionality(0)
                                 as MapFunctionality;
    localMapResource = (MapResourceLocal)agsMapFunctionality.Resource;

    // 通过 ArcObjects 删除动态添加的图层
```

```
        IMapServerObjects mapServerObjects = localMapResource.MapServer
                        as IMapServerObjects;
        IMap map = mapServerObjects.get Map(localMapResource.DataFrame);
        ILayer workingLayer = null;
        IEnumLayer enumLayer = map.get Layers(null, true);
        ILayer loopLayer = null;
        while ((loopLayer = enumLayer.Next()) != null) {
            if (loopLayer.Name == "Canada") {
                workingLayer = loopLayer;
            }

        }
        map.DeleteLayer(workingLayer);

        // 刷新服务器对象，响应删除图层
        localMapResource.RefreshServerObjects();

        Toc1.Refresh();
        Map1.CallbackResults.CopyFrom(Toc1.CallbackResults);

        Map1.Refresh();
        callbackResponse = Map1.CallbackResults.ToString();
    }
```


在删除图层的方法中，首先得到本地资源对象，从该对象中得到服务器对象，从该对象中得到地图对象，从地图对象中删除加入的图层。然后调用本地资源对象的 RefreshServerObjects 方法，刷新服务器对象。最后刷新 Toc 与地图。

编译并运行程序，通过“增加按钮图层“与”删除图层”两个按钮分别添加与删除图层。运行效果如图 5.16 所示。

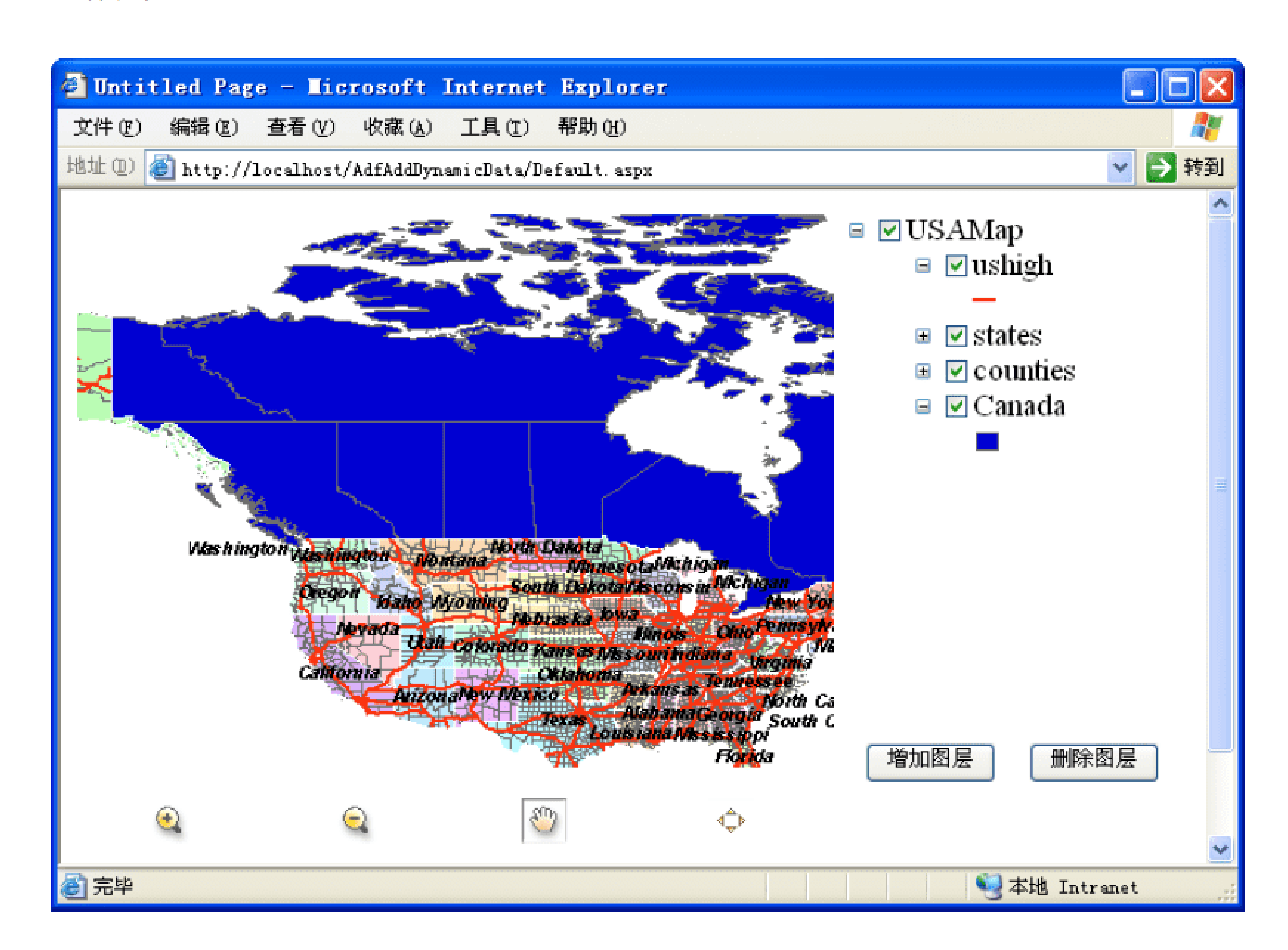


图 5.16　通过本地地图资源对象添加图层

5.4.3 地图资源的功能

地图资源可提供绘图功能、查询功能、在 Toc 中显示功能、地图切片功能以及比例尺显示功能。

1. 绘图功能

使用地图资源最多的当然是地图控件。地图控件使用每个地图资源创建一个相应的绘图功能。绘图功能类的 Resource 属性是抽象的 IMapResource 接口的引用，而 MapResource 属性则对应特定地图资源类的引用。地图控件提供了两个方法来得到功能对象，一个是 GetFunctionalities 方法，返回一组 IMapFunctionality 对象；另一个是 GetFunctionality 方法，根据参数的值返回一个 IMapFunctionality 对象。

从开发者的角度来看，一般在以下三种情况时会使用绘图功能。一是得到地图资源的元数据，例如地图中包含的图层名、图层 ID、比例尺以及空间参考等，二是绘制地图，三是使用地图资源特定数据源专有的功能。

表 5-7 列出了地图功能类主要的属性与方法。

表 5-7 地图功能类主要的属性与方法

属性或方法	说明
SpatialReference	表示地图资源的空间参考
DisplaySettings	绘制地图时使用的一些属性，例如透明程序、背景颜色等
MaintainsState	该属性决定是否使用功能或资源来维护状态
GetLayers	得到地图资源中所有图层的名称与 ID
Get\SetLayerVisibility()	改变图层的可见性
GetScale()	返回当前显示比例尺
DrawExtent()	生成一张地图图片

IMapFunctionality 作为公有 API 中的接口，在使用时不需要知道底层使用的具体的数据源类型。还要注意的是，在地图控件中，将一绘图功能作为一个图层来使用，但是绘图功能自身可能包含了许多图层。在本节后面的实例中将演示如何得到地图控件中所有的绘图功能，以及每个绘图功能所包含的图层。

绘图功能一个重要的特点是，Web ADF 使用它来维护用户会话过程中的状态。绘图功能中存储了生成地图图片的对象的引用，例如可见图层、地图属性等。Web ADF 以无状态的方式使用数据源，这意味着数据源是不保存状态。Web ADF 是在 Web 层保存状态的，因此，可以在程序运行期间改变地图功能的属性，并在当前会话过程中维护该状态。例如，当在地图功能中加入服务器端的图形后（例如在 ArcGIS Server 数据源的 MapDescription 中加入 GraphicElements，即 4.4.3 节中介绍的方法），在与服务器数据源请求过程中会一直保留，而不需要每次请求时重新加入。

下面通过一个实例来介绍如何使用公有 API 中的地图功能。该实例实现的功能有些类似 Toc 控件的功能，可用来显示各资源及其图层的可见性，并可控制它们的可见性。

在 Visual Studio 2005 中，利用 File 菜单的 New Web Site 命令，创建一新的 Web 站点，命名

为 SetLayerVisibility。

从工具箱的 ArcGIS Web Controls 选项卡中，在 Default.aspx 页面中增加一地图资源管理器控件、一地图控件与一工具栏控件。在地图资源控件中先后加入 USAMap 与 NorthAmericaMap 资源，并将 NorthAmericaMap 资源的可见属性设置为 false。通过工程右键菜单的 Add ArcGIS Identity 命令加入身份验证信息。用 4.1.3 节介绍的方法加入其他属性设置，使几个控件能联动。

从工具箱的 HTML 选项卡中，在 Default.aspx 页面中增加一 Div 控件，ID 设置为 layerListDiv，放置在地图控件的右侧。

本实例用到了一个 JavaScript 文件、一样式文件与一些图标文件，需要读者将本书源文件中 SetLayerVisibility 目录下的 TreeRes 文件夹拷贝到读者创建的工程的目录中。

在 Default.aspx 文件的<head>与</head>之间加入如下代码，引用上述样式文件与 JavaScript 文件：

```
<link rel="stylesheet" type="text/css" href="TreeRes/XMLSelTree.css" />
<script language="javascript" src="TreeRes/SimpleSelTree.js"
   type = "text/javascript"/>
```

由于希望程序启动后立即显示地图资源及其图层状态信息，因此在 Default.aspx 文件需要加入如下 JavaScript 语句：

```
<script type="text/javascript" language="javascript">
   function window.onload() {
      GetMapLayers();
   }
</script>
```

该代码表示页面加载时执行 GetMapLayers 函数，该函数是我们自定义的。在<head>与</head>之间加入该函数的代码，如下所示：

```
<script  language="javascript" type="text/javascript">
   function GetMapLayers() {
      var message = 'ActiveType=GetMapLayers';
      var context = 'Page1';
      <%=getMapLayersCallBack%>
   }
</script>
```

上述代码执行时调用 getMapLayersCallBack 代表的回调代码。

在 Default.aspx.cs 文件中，首先将类的声明代码行后面加入对 ICallbackEventHandler 接口的实现，然后在类中加入如下两个字段：

```
public string getMapLayersCallBack; // 客户端函数的引用
private string callbackArg; // 回调事件的结果
```

在 Page_Load 方法中加入如下代码，获取一个对客户端函数的引用：

```
protected void Page Load(object sender, EventArgs e) {
   getMapLayersCallBack = ClientScript.GetCallbackEventReference(this,
      "message", "processCallbackResult", "context", "postBackError", true);
}
```

加入 ICallbackEventHandler 接口要求的两个方法的实现代码，如下所示：

```
void ICallbackEventHandler.RaiseCallbackEvent(string eventArgument) {
    callbackArg = eventArgument;
}

string ICallbackEventHandler.GetCallbackResult()
{
    // 将传入参数依据&分割符分到 querystring 变量中
    Array keyValuePairs = callbackArg.Split("&".ToCharArray());
    NameValueCollection queryString = new NameValueCollection();
    string[] keyValue;
    string response = "";
    if (keyValuePairs.Length > 0) {
        for (int i = 0; i < keyValuePairs.Length; i++) {
            keyValue =
               keyValuePairs.GetValue(i).ToString().Split("=".ToCharArray());
            queryString.Add(keyValue[0], keyValue[1]);
        }
    }
    else {
        keyValue = callbackArg.Split("=".ToCharArray());
        if (keyValue.Length > 0)
            queryString.Add(keyValue[0], keyValue[1]);
    }

    // 针对参数中指定的 ActiveType 不同执行不同操作
    string controlType = queryString["ActiveType"];
    switch (controlType) {
        case "GetMapLayers":
            response = GisFunctionality.GetMapLayers(Map1);
            break;
        default:
            break;
    }

    return response;
}
```

上述代码与前面一些实例的很类似，实现获取地图资源及其图层信息的是 GisFunctionality 类的 GetMapLayers 方法。

在工程中加入 ASP.NET 文件夹 App_Code，在其中新增加 GisFunctionality 类。在该类中首先加入如下一些命名空间的引用：

```
using System.Collections;
using ESRI.ArcGIS.ADF.Web.UI.WebControls;
using ESRI.ArcGIS.ADF.Web.DataSources;
```

GetMapLayers 方法要实现的功能是，依据地图控件地图资源及其图层，构造一类似 Toc 控件内容的字符串，将该字符串作为 layerListDiv 的内容显示。代码如下：

```
    public static string GetMapLayers(Map map) {
        System.Text.StringBuilder strBld = new System.Text.StringBuilder();

        IEnumerable gfc = map.GetFunctionalities();
        foreach (IGISFunctionality gisfunc in gfc) {
            IMapFunctionality mf = gisfunc as IMapFunctionality;
            if (mf == null)
                continue;

            strBld.Append("<div class='clsItem' type='branch'>");
            strBld.Append(@"<img type='img'
               onclick='MouseClick(this)' src='TreeRes/Images/NodeImg1.gif' />");
            bool resourceVisible = mf.DisplaySettings.Visible;
            string checkedValue = "";
            if (resourceVisible)
                checkedValue = "checked='checked'";
            string inputBox = string.Format("<input id='{0}' type='checkbox'
                                 onclick='ResourceVisible(this)' {1}/>",
                                 mf.Resource.Name, checkedValue);
            strBld.Append(inputBox);

            string head = @"<span class='clsLabel'
                                 type='label'
                                 onmouseover='NodeMouseOver(this);'
                                 onmouseout='NodeMouseOut(this)'
                                 style='position: relative; left: -4px;'>
                           <span class='Link'>";
            strBld.Append(head);
            strBld.Append(mf.Resource.Name);
            strBld.Append("</span></span><div class='hide' type='container'>");

            string[] layerIDs;
            string[] layerNames;
            mf.GetLayers(out layerIDs, out layerNames);
            for(int i=0; i<layerIDs.Length; i++) {
                strBld.Append("<div class='clsItem' type='leaf'>");
                string checkedLayer = "";
                bool layerVisible = mf.GetLayerVisibility(layerIDs[i]);

                if (layerVisible)
                    checkedLayer = "checked='checked'";
                string layerInputBox = string.Format("<input id='{0}'
                          type='checkbox'
                          onclick=\"LayerVisible(this, '{1}')\" {2}/>",
                          layerIDs[i], mf.Resource.Name, checkedLayer);
                strBld.Append(layerInputBox);
                strBld.Append(@"<span class='clsLabel' type='label'
                                 onmouseover='NodeMouseOver(this);'
                                 onmouseout='NodeMouseOut(this);'
                                 style='position:relative;left:2px;' >");
```

```
                strBld.Append(layerNames[i]);
                strBld.Append("</span></div>");
            }

            strBld.Append("</div></div>");
        }

        CallbackResult cr = new CallbackResult("div", "layerListDiv",
            "innercontent", strBld.ToString());
        map.CallbackResults.Add(cr);
        return map.CallbackResults.ToString();
    }
```

在上面的代码中，首先调用地图控件的 GetFunctionalities 方法得到所有的功能，然后对每个功能进行循环。

在循环的代码中，只对绘图功能进行处理。调用绘图功能的 GetLayers 方法得到该功能中所有图层及其 ID，然后对图层进行循环，得到每个图层的名称，并用 GetLayerVisibility 方法得到图层的可见性，依据可见性构造 HTML 控件。

代码最后使用 CallbackResult 方法将构造好的字符串指定显示到类型为“Div”，ID 为“layerListDiv”的控件中。

编译并运行程序，运行效果如图 5.17 所示。

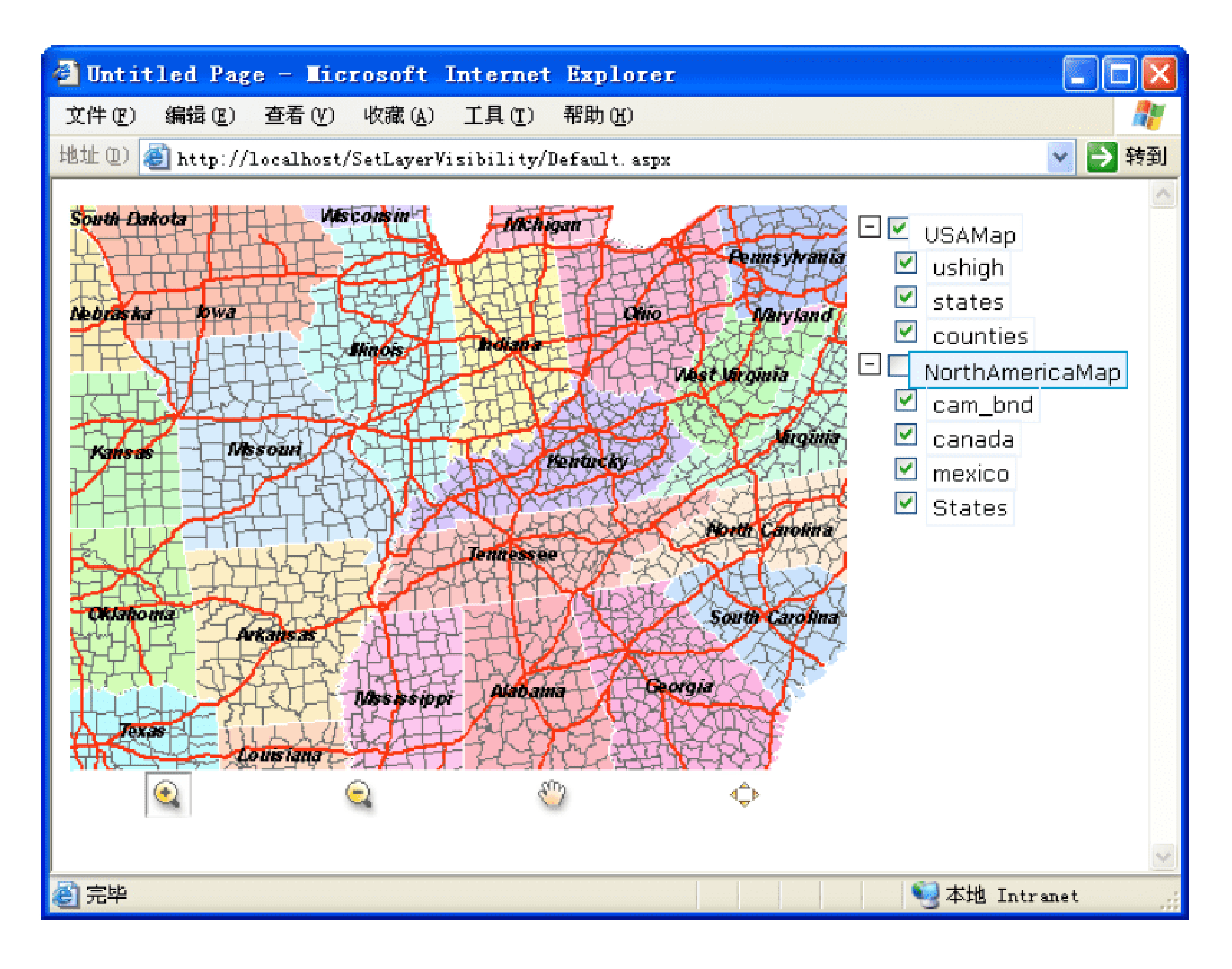


图 5.17　得到地图资源及其图层的名称与可见性

在上面的代码中，实现了得到地图资源及其图层的名称与可见性，下面要完成的是控制地图资源及图层的可见性，在前面的代码中，我们已经为每个复选框指定了 onclick 事件响应函数，对于资源级别的复选框使用的是 ResourceVisible 函数，对于图层级别使用的是 LayerVisible。

先来实现控制地图资源的可见性。在 Default.aspx 的 JavaScript 代码段中，加入如下代码，以

响应用户选择资源级别的复选框的操作：

```
function ResourceVisible(checkBox) {
    var message = 'ActiveType=SetResource&ResourceName=';
    message += checkBox.id;
    message += "&Visible=" + checkBox.checked;
    var context = 'Page1';
    <%=resourceCallBack%>
}
```

然后切换到 Default.aspx.cs 文件中，加入一个字段：

```
public string resourceCallBack;
```

在 Page_Load 方法中加入如下代码，获取一个对客户端函数的引用：

```
resourceCallBack = ClientScript.GetCallbackEventReference(this,
    "message", "processCallbackResult", "context", "postBackError", true);
```

然后在 GetCallbackResult 方法的 switch 语句中加入如下 case 语句段，用于通过调用 SetResourceVisibility 方法实现设置地图资源的可见性：

```
case "SetResource":
    response = GisFunctionality.SetResourceVisibility(Map1,
        queryString["ResourceName"], queryString["Visible"]);
    break;
```

SetResourceVisibility 方法的代码如下：

```
public static string SetResourceVisibility(Map map, string resourceName,
string visibled) {
    IGISFunctionality gisfunc = map.GetFunctionality(resourceName);
    if (gisfunc == null)
        return "";

    IMapFunctionality mf = gisfunc as IMapFunctionality;
    if (visibled == "true")
        mf.DisplaySettings.Visible = true;
    else
        mf.DisplaySettings.Visible = false;

    RefreshMap(map, resourceName);

    return map.CallbackResults.ToString();
}
```

代码很简单，首先通过地图控件的 GetFunctionality 方法，得到用户选择的地图资源的功能，然后通过设置该功能对象的 DisplaySettings 属性的 Visible 属性，从而控制地图资源的可见性，然后调用 RefreshMap 方法刷新地图，最后返回地图回调对象中保存的回调内容。

RefreshMap 方法的代码如下，用于根据不同属性的设置调用不同的方法来刷新地图：

```
public static void RefreshMap(Map map, string resourceName) {
    if (map.ImageBlendingMode == ImageBlendingMode.WebTier) {
        map.Refresh();
```

```
        }
        else if (map.ImageBlendingMode == ImageBlendingMode.Browser) {
            map.RefreshResource(resourceName);
        }
    }
```

编译并运行程序，读者现在应该可以控制两个地图资源的可见性了。下面要实现的是控制图层的可见性。

在 Default.aspx 的 JavaScript 代码段中，加入如下代码，响应用户选择图层级别的复选框的操作：

```
    function LayerVisible(checkBox, resourceName) {
        var message = 'ActiveType=SetLayer&ResourceName=';
        message += resourceName;
        message += "&LayerID=" + checkBox.id;
        message += "&Visible=" + checkBox.checked;
        var context = 'Page1';
        <%=layerCallBack%>
    }
```

然后切换到 Default.aspx.cs 文件中，加入一个字段：

```
    public string layerCallBack;
```

在 Page_Load 方法中加入如下代码，获取一个对客户端函数的引用：

```
    layerCallBack= ClientScript.GetCallbackEventReference(this,
        "message", "processCallbackResult", "context", "postBackError", true);
```

然后在 GetCallbackResult 方法的 switch 语句中加入如下 case 语句段，用于通过调用 SetLayerVisibility 方法实现设置地图资源的可见性：

```
    case "SetLayer":
        response = GisFunctionality.SetLayerVisibility(
                        Map1, queryString["ResourceName"],
                        queryString["LayerID"], queryString["Visible"]);
        break;
```

SetLayerVisibility 方法的代码如下：

```
    public static string SetLayerVisibility(Map map, string resourceName,
    string layerId, string visibled)
    {
        IGISFunctionality gisfunc = map.GetFunctionality(resourceName);
        if (gisfunc == null)
            return "";
        IMapFunctionality mf = gisfunc as IMapFunctionality;
        if (visibled == "true")
            mf.SetLayerVisibility(layerId, true);
        else
            mf.SetLayerVisibility(layerId, false);

        RefreshMap(map, resourceName);
```

```
        return map.CallbackResults.ToString();
    }
```

在上述代码中，通过绘图功能的 SetLayerVisibility 方法设置图层的可见性。

编译并运行程序。现在我们可以不需要 Toc 控件，而使用自定义的代码来控制地图中图层的可见性了。程序运行效果如图 5.18 所示。

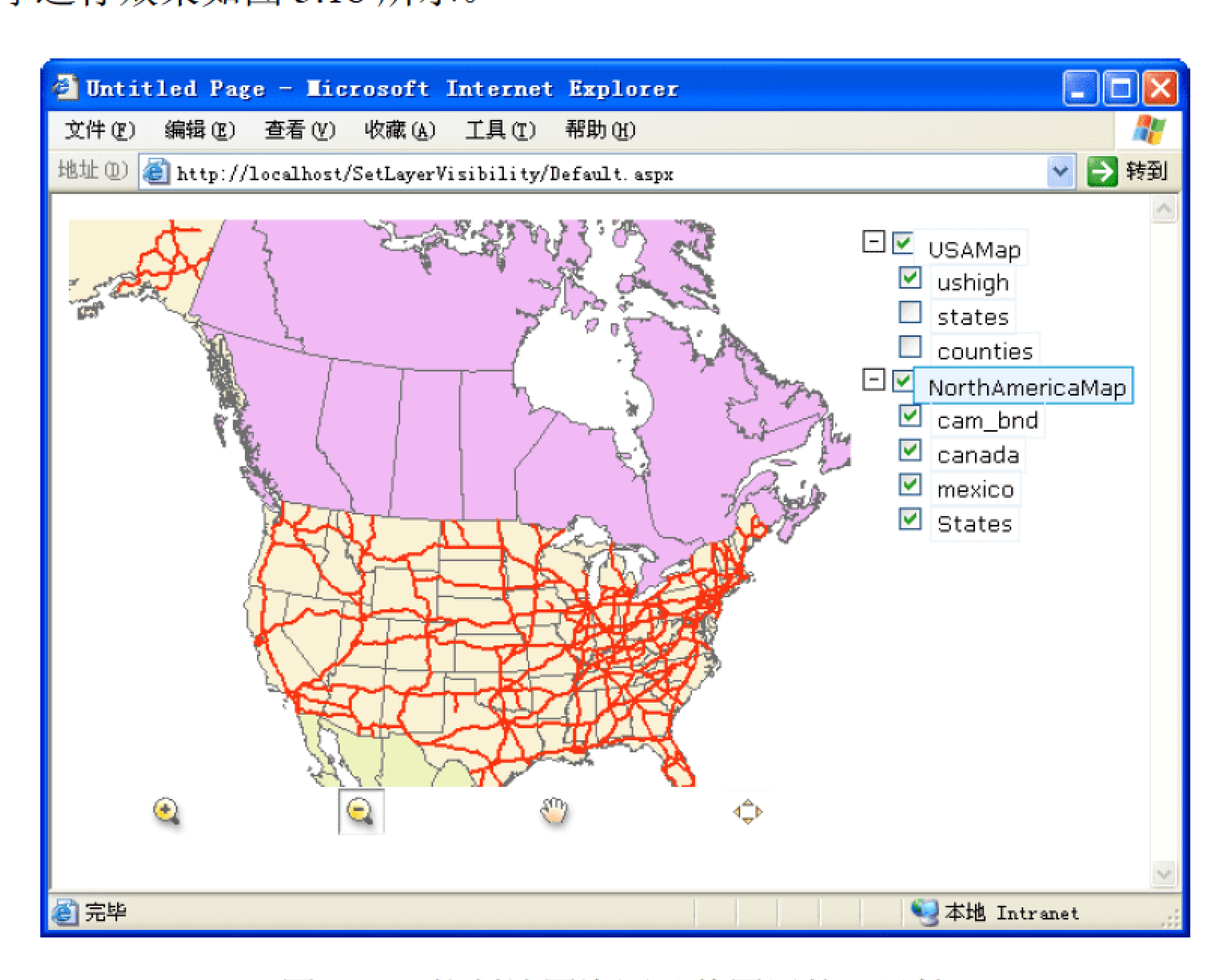


图 5.18 控制地图资源及其图层的可见性

2. 查询功能

IQueryFunctionaltiy 接口提供对数据源的要素数据执行空间与属性的查询功能。要获取该功能，必须使用地图资源的 CreateFunctionality 方法来创建。在大多数情况下，只需要使用公有 API 中的 IQueryFunctionaltiy 接口即可实现查询功能，只有在很少的特殊情况下才需要使用数据源特有的查询功能。

表 5-8 列出了 IQueryFunctionaltiy 接口提供的主要方法。

表 5-8 IQueryFunctionaltiy 接口的主要方法

方法	说明
GetQueryableLayers	得到能用于查询的要素图层的名称及 ID 数组。地图中有些图层是不能查询的，例如栅格图层
GetFields	返回地图资源中某图层的字段数组
Find	对地图资源中的一个或多个图层查询某个值，返回一 DataTable 数组
Identify	对地图资源中的一个或多个图层执行空间查询，即通过图形查询属性
Query	对地图资源中的一个图层执行空间与/或属性查询

GetQueryableLayers 与 GetFields 方法用于在地图资源中找到能用于查询的信息。Find、Identify 与 Query 方法返回的是 ADO.NET 中常用的数据对象 System.Data.DataTable 类型。如果查询返回

Web ADF 的几何图形，可以在图形资源的图形图层中使用。

在第 4 章中介绍了 Identify 与 Query 两个方法，下面通过一个实例来介绍 Find 方法的使用。该实例要实现的功能是全文搜索，在所有图层的所有字段中搜索用户的输入。

在 Visual Studio 2005 中，利用 File 菜单的 New Web Site 命令，创建一新的 Web 站点，命名为 FullSearch。

同样，本实例用到了一个样式文件与一些图标文件，需要读者将本书源代码文件中 SetLayerVisibility 目录下的 css 与 Images 文件夹拷贝到读者创建的工程的目录中。

从工具箱的 ArcGIS Web Controls 选项卡中，在 Default.aspx 页面中增加一地图资源管理器控件、一地图控件与一工具栏控件。在地图资源控件中先后加入一图形图层、USAMap 与 NorthAmericaMap 资源。并通过工程右键菜单的 Add ArcGIS Identity 命令加入身份验证信息。用 4.1.3 节介绍的方法加入其他属性设置，使几个控件能联动。

从工具箱的 HTML 选项卡中，在 Default.aspx 页面中地图控件的右侧增加一文本框控件、一按钮控件与一 Div 控件，将其 id 分别设置为 queryCondition、Query 与 resultDiv。

将按钮控件的 onclick 事件设置为 FullQuery()函数，将用该函数来响应用户的查询。在 Default.aspx 的<head></head>中加入如下代码，用于设置样式文件与响应用户的查询操作：

```
<link href="css/style1.css" type="text/css" rel="stylesheet"/>
<script  language="javascript" type="text/javascript">
   function FullQuery() {
      var condition = document.getElementById("queryCondition");
      var message = 'ActiveType=FullQuery&Condtion=';
      message += condition.value;
      var context = 'Page1';
      <%=fullQueryCallBack%>
   }
</script>
```

切换到 Default.aspx.cs 文件中，修改 _Default 类的声明为如下代码，增加对 ICallbackEventHandler 接口的实现。

```
public partial class _Default : System.Web.UI.Page, ICallbackEventHandler
```

在_Default 类中加入如下两个字段：

```
public string fullQueryCallBack;
private string callbackArg;
```

在 Page_Load 方法中加入如下代码：

```
protected void Page_Load(object sender, EventArgs e) {
   fullQueryCallBack = ClientScript.GetCallbackEventReference(this,
      "message", "processCallbackResult", "context", "postBackError", true);
}
```

然后要完成的是回调接口需要实现的两个方法，代码如下：

```
void ICallbackEventHandler.RaiseCallbackEvent(string eventArgument) {
   callbackArg = eventArgument;
}
```

```
string ICallbackEventHandler.GetCallbackResult() {
    // 将传入参数依据&分割符分到 querystring 变量中
    Array keyValuePairs = callbackArg.Split("&".ToCharArray());
    NameValueCollection queryString = new NameValueCollection();
    string[] keyValue;
    string response = "";
    if (keyValuePairs.Length > 0) {
        for (int i = 0; i < keyValuePairs.Length; i++) {
            keyValue =
              keyValuePairs.GetValue(i).ToString().Split("=".ToCharArray());
            queryString.Add(keyValue[0], keyValue[1]);
        }
    }
    else {
        keyValue = callbackArg.Split("=".ToCharArray());
        if (keyValue.Length > 0)
            queryString.Add(keyValue[0], keyValue[1]);
    }

    // 针对参数中指定的 ActiveType 不同执行不同操作
    string controlType = queryString["ActiveType"];
    switch (controlType) {
        case "FullQuery":
            response = GisFunctionality.GetResourceContent(Map1,
                                    queryString["Condtion"]);
            break;
        default:
            break;
    }

    return response;
}
```

上面的代码在判断用户执行的是全文查询操作时，调用 GisFunctionality 类的静态方法 GetResourceContent 来执行查询。

在工程中加入 ASP.NET 的文件夹 APP_CODE，并在其中加入 Gourcer Wew 类。

在 Gourcer Wew 类中加入如下命名空间的引用：

```
using System.Collections;
using ESRI.ArcGIS.ADF.Web.UI.WebControls;
using ESRI.ArcGIS.ADF.Web.DataSources;
using ESRI.ArcGIS.ADF.Web;
```

先在 GetResourceContent 方法中加入如下代码：

```
public static string GetResourceContent(Map map, string condition)
{
    System.Text.StringBuilder sbContent = new System.Text.StringBuilder();
    sbContent.Append("<table border=0 cellpadding=0
                        cellspacing=1 class=list-line>");
```

```
    sbContent.Append("<tr><td class=listbg>资源名</td>
                     <td class=listbg>图层名</td>
                     <td class=listbg>操作</td></tr>");
}
```

由于执行的是全文搜索，查询结果可能分布在不同的地图资源中，还可能在同一个图层中有几个记录满足条件。因此需要一个表格将查询结果按资源与图层列出来。在上面的代码中，先构造了一个表格的框架。

接着要实现的是对地图资源进行查询。在 GetResourceContent 方法中加入如下代码：

```
IEnumerable func_enum = map.GetFunctionalities();
foreach (IGISFunctionality gisfunctionality in func_enum)
{
    IGISResource gisresource = gisfunctionality.Resource;
    bool supported =
      gisresource.SupportsFunctionality(typeof(IQueryFunctionality));

    if (!supported)
        continue;

    IQueryFunctionality qfunc;
    qfunc =
      gisresource.CreateFunctionality(typeof(IQueryFunctionality), null)
      as IQueryFunctionality;
    if (qfunc == null)
        continue;
}
```

由于地图中包含几个资源，因此先通过地图控件的 GetFunctionalities 得到所有的功能，然后对每个功能进行循环查询。在循环中首先判断资源是否支持查询，如果不支持，则进行下一个循环。如果支持查询，则调用资源的 CreateFunctionality 方法，创建查询功能，如果不成功，则进行下一个循环。

在 foreach 循环中再加入如下代码：

```
string[] lids;
string[] lnames;
qfunc.GetQueryableLayers(null, out lids, out lnames);
if (lids.Length < 1)
    continue;
```

在上述代码中，通过查询功能的 GetQueryableLayers 方法得到资源中能执行查询的图层名及其对应 ID 数组。由于是图形图层资源，没有图层可返回，因此需要进行数组大小的判断。

在 foreach 循环中再加入如下代码，构造查询参数：

```
FindParameters findParam = new FindParameters();
findParam.FindOption = FindOption.AllLayers;
findParam.FindString = condition;
findParam.MaxRecords = 50;
for (int i = 0; i < lids.Length; i++) {
    string[] fields = qfunc.GetFields(null, lids[i]);
```

```
        findParam.LayersAndFields.Add(lids[i], fields);
    }
```

上述代码得到对图层数组进行循环，在其中调用查询功能的 GetFields 方法得到每个图层的所有字段，并将图层及其字段加入到查询参数对象的 LayersAndFields 属性中。

在 foreach 循环中再加入如下代码：

```
    System.Data.DataTable[] dtable = qfunc.Find(null, findParam);
    if (dtable == null)
        continue;
    if (dtable.Length == 0)
        continue;
```

上述代码调用查询功能的 Find 方法，根据查询参数对象，对整个地图资源进行查询，并返回一数据表数组，如果没有找到匹配记录，则返回结果为 null 或数组长度为 0，则进行对象下一个资源的查询。

上述代码已经将查询结果保存在了数据表数组中了，下面要完成的是根据这个结果，填充表格。

在 foreach 循环中再加入如下代码：

```
    for (int i = 0; i < dtable.Length; i++) {
        string formatStr = "<tr><td class=listbg>{0}</td>
                            <td class=listbg>{1}</td>";
        formatStr += "<td class=listbg>
                          <A href=\"javascript: void Position('{2}', '{3}',
                          '{4}', '{5}')\">定位<A></td></tr>";
        string primaryFieldName = PrimaryFieldName(dtable[i]);
        for (int j = 0; j < dtable[i].Rows.Count; j++) {
            string rowPrimaryFieldValue =
                dtable[i].Rows[j][primaryFieldName].ToString();
            string row = string.Format(formatStr,
                          gisresource.Name, dtable[i].TableName,
                          gisresource.Name, dtable[i].TableName,
                          primaryFieldName, rowPrimaryFieldValue);
            sbContent.Append(row);
        }

    }
```

上述代码对数据表数组进行循环，在其中对数据表的记录集进行循环，得到该记录的主键列的值，然后根据该值、资源名称以及图层名称，构造表格中的一行 HTML 代码，在该代码中包含一个超链接，我们将利用该超链接在地图上定位到该行对应的要素。

在 foreach 循环之后加入如下代码：

```
    sbContent.Append("</table>");

    CallbackResult cr = new CallbackResult("div", "resultDiv",
                         "innercontent", sbContent.ToString());
    map.CallbackResults.Add(cr);
```

```
    return map.CallbackResults.ToString();
```

上述代码利用 CallbackResult 类将我们刚才生成的表作为 resultDiv 控件的内容显示，并将该对象加入到地图控件的 CallbackResults 属性中，最后将该属性作为字符串返回。

编译并运行程序，在查询条件文本框中输入加拿大或美国的州名，例如 Saskatchewan、Minnesota 与 Alaska 等。程序运行效果如图 5.19 所示。

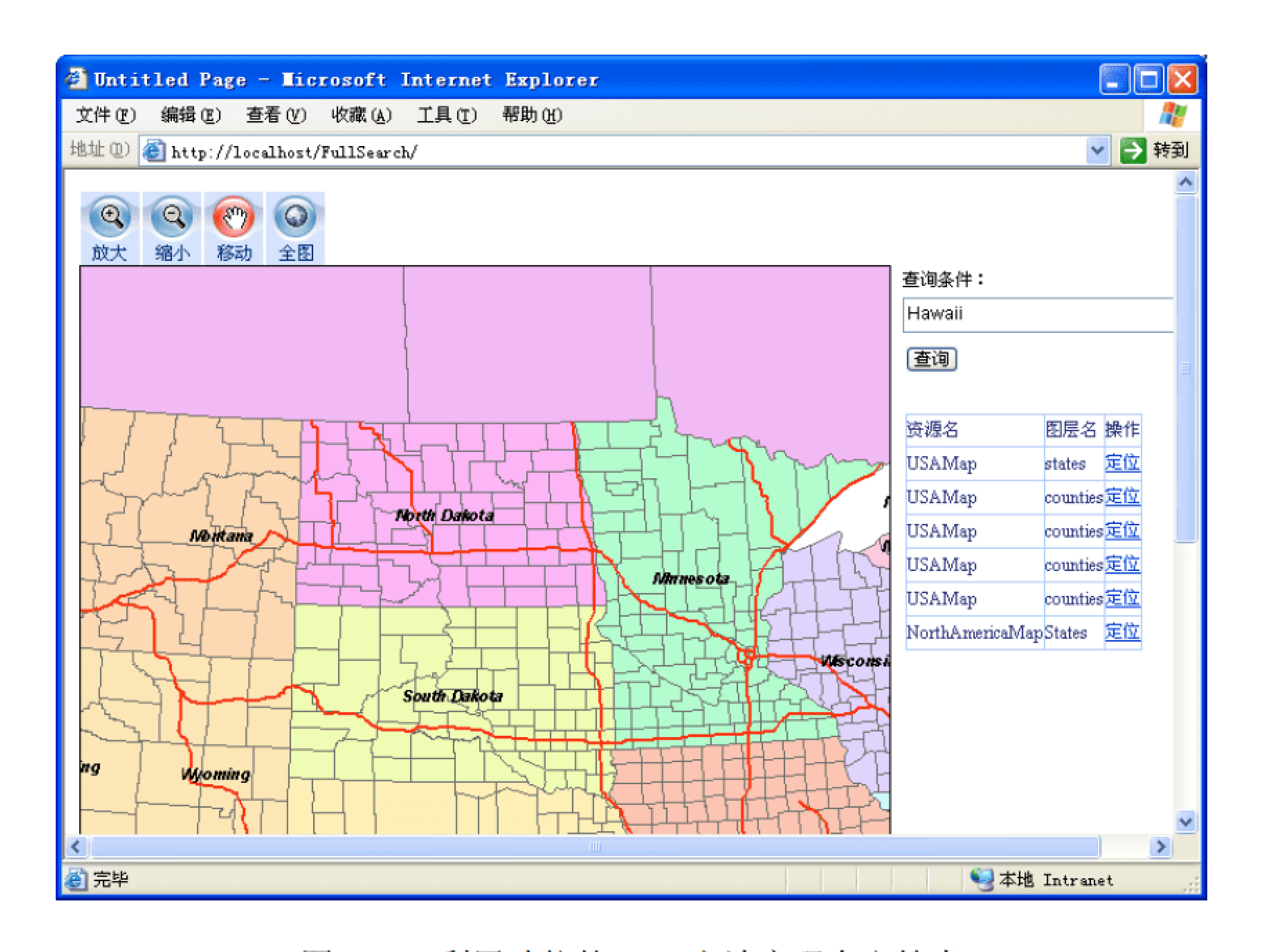


图 5.19　利用功能的 Find 方法实现全文搜索

接着要实现的是在地图上定位查询到的结果。

在 Default.aspx 的<head>与</head>之间的 JavaScript 函数处加入 Position 函数的代码，如下所示：

```
function Position(resourceName, layerName, fieldName, fieldValue) {
    var message = 'ActiveType=Position&ResourceName=';
    message += resourceName;
    message += "&LayerName=" + layerName;
    message += "&FieldName=" + fieldName;
    message += "&FieldValue=" + fieldValue;
    var context = 'Page1';
    <%=positionCallBack%>
}
```

切换到 Default.aspx.cs 文件中，在类中首先加入如下一字段：

```
public string positionCallBack;
```

在 Page_Load 方法中加入如下代码，为 positionCallBack 赋值：

```
positionCallBack = ClientScript.GetCallbackEventReference(this,
    "message", "processCallbackResult", "context", "postBackError", true);
```

在 GetCallbackResult 方法的 switch 中加入如下 case 代码段：

```
case "Position":
    response = GisFunctionality.Position(Map1, queryString["ResourceName"],
                                          queryString["LayerName"],
                                          queryString["FieldName"],
                                          queryString["FieldValue"]);
    break;
```

上述代码调用GisFunctionality类的Position方法实现定位选择结果。该方法的代码如下：

```
public static string Position(Map map, string resourceName,
    string layerName, string fieldName, string fieldValue) {
    IGISFunctionality gisfunc = map.GetFunctionality(resourceName);
    if (gisfunc == null)
        return "";

    IGISResource gisresource = gisfunc.Resource;
    bool supportquery =
       gisresource.SupportsFunctionality(typeof(IQueryFunctionality));
    if (!supportquery)
        return "";

    IQueryFunctionality qfunc;
    qfunc =
       gisresource.CreateFunctionality(typeof(IQueryFunctionality), null)
       as IQueryFunctionality;

    SpatialFilter spatialfilter = new SpatialFilter();
    spatialfilter.ReturnADFGeometries = false;
    spatialfilter.MaxRecords = 1000;
    spatialfilter.WhereClause = fieldName + "=" + fieldValue;

    string layerId = GetQueryableLayerIDFromName(qfunc, layerName);
    System.Data.DataTable datatable =
       qfunc.Query(null, layerId, spatialfilter);
    if (datatable == null)
        return "";

    return HighlightShow(map, datatable);
}
```

实现代码很简单，主要代码是调用查询功能的 Query 方法在指定图层中查询。由于 Query 方法需要的是图层的 ID，而参数传进来的是图层名称，因此在调用 Query 方法之前，调用了 GetQueryableLayerIDFromName 方法根据图层名得到图层的 ID。该方法及其支持方法的代码如下：

```
public static string GetQueryableLayerIDFromName(IQueryFunctionality qfunc
string layerName) {
string[] layerIDs;
string[] layerNames;
qfunc.GetQueryableLayers(null, out layerIDs, out layerNames);
```

```
    return GetLayerID(layerIDs, layerNames, layerName);
    }

    public static string GetLayerID(string[] layerIDs,
    string[] layerNames, string findLayerName) {
    for (int i = 0; i < layerIDs.Length; i++) {
    if (layerNames[i] == findLayerName)
    return layerIDs[i];
    }

    return "";
    }
```

高亮显示的方法，即 HighlightShow，完全同 4.5 节中 AttributeQuery 工程中同名方法，因此为节省篇幅，这里不再给出。

编译并运行程序，查看程序运行结果。运行效果如图 5.20 所示。

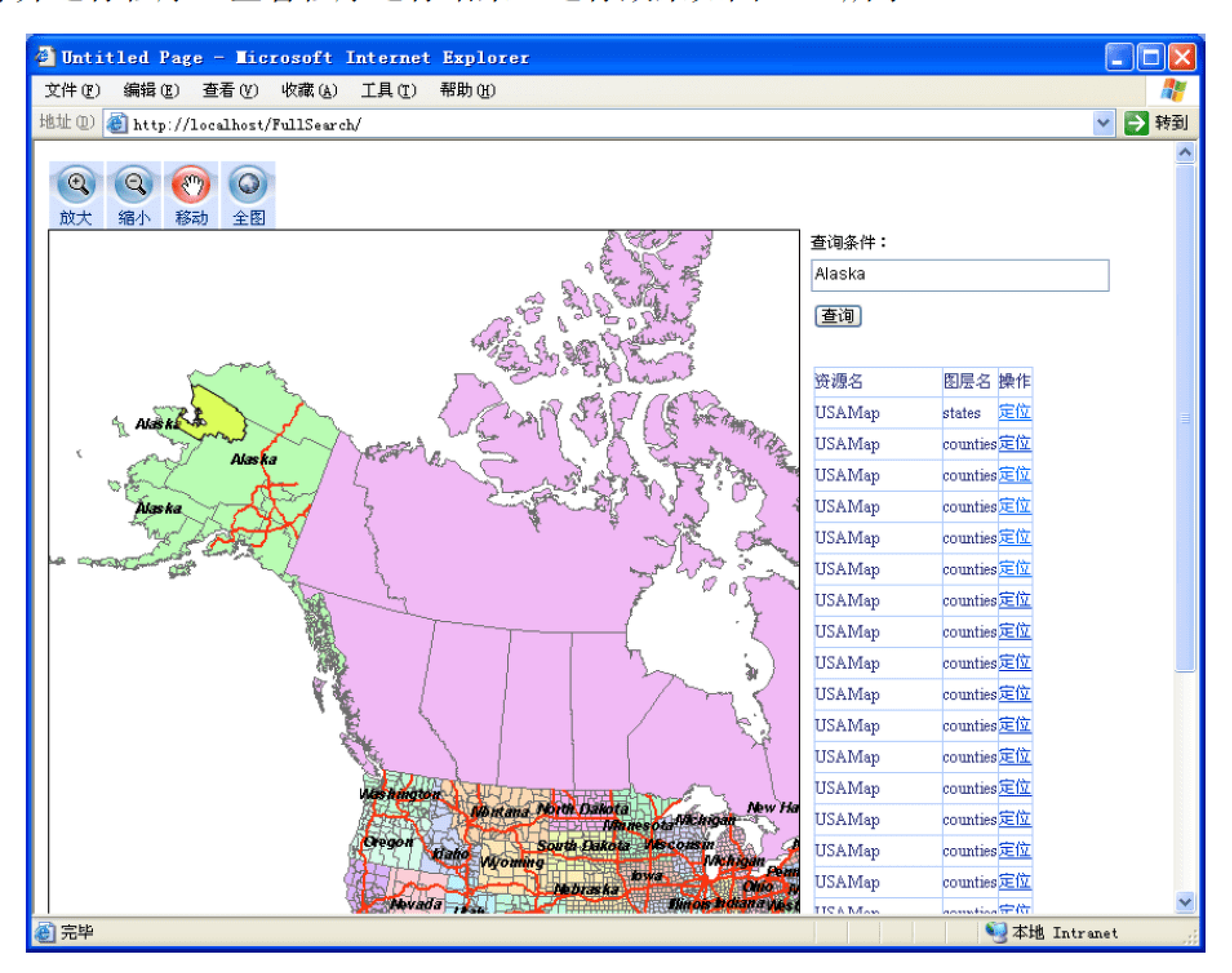


图 5.20　定位查询结果

3．其他功能

虽然对于程序开发人员来说，编码过程中用得最多的是绘图功能与查询功能，但是其他几个功能，特别是 Toc 功能，也是整个应用程序不可缺少的部分。

通过 MapTocFunctionality 可访问 Toc 控件中数据。每个资源对应一个 TocDataFrame 对象，而每个图层对应一个 TocLayer 对象。可以通过绘图功能的 GetLayerVisibility 与 SetLayerVisibility 方法来改变 Toc 中图层的可见性。

❑　TileFunctionality 能够利用事先创建的地图切片文件，而不是每次都使用绘图功能创建地

图图片。Web ADF 自身只针对 ArcGIS Server 数据源提供该功能。通常，只有在使用自定义数据源时才需要使用该功能。

ScaleBarFunctionality 能够为应用程序生成一个比例尺。

下面我们通过一实例来演示如何使用 Toc 功能。要实现的功能是在页面上显示图例。

在 Visual Studio 2005 中，利用 File 菜单的 New Web Site 命令，创建一新的 Web 站点，命名为 FullSearch。

本实例用到的样式文件与一些图标文件，需要读者将本书所附源代码文件中 SetLayerVisibility 目录下的 css 与 Images 文件夹拷贝到读者创建的工程的目录中。

从工具箱的 ArcGIS Web Controls 选项卡中，在 Default.aspx 页面中增加一地图资源管理器控件与一地图控件。在地图资源控件中 NorthAmericaMap 资源。并通过工程右键菜单的 Add ArcGIS Identity 命令加入身份验证信息。用 4.1.3 节介绍的方法加入其他属性设置，使几个控件能联动。

从工具箱的 HTML 选项卡中，在 Default.aspx 页面中地图控件的右侧增加一 Image 控件，将其 id 分别设置为 legend，将其高与宽都设置为 200。我们将在该控件中显示图例。

由于我们希望在页面加载时便显示图例。因此需要在 window 的 onload 事件中执行回调，利用 Toc 功能绘制一图例图象。在 Default.aspx 的代码的最后加入如下代码：

```
<script type="text/javascript" language="javascript">
    function window.onload() {
        GetLegend();
    }

    function GetLegend() {
        var message = 'ActiveType=GetLegend';
        var context = 'Page1';
        <%=getLegendCallBack%>
    }
</script>
```

切换到 Default.aspx.cs 文件中，首先在_Default 类中加入如下两个字段：

```
public string getLegendCallBack; // 客户端脚本段
private string callbackArg; // 返回给客户端的内容
```

在 Page_Load 方法中加入如下代码：

```
protected void Page Load(object sender, EventArgs e)
{
    // 利用 GetCallbackEventReference 方法生成客户端脚本
    getLegendCallBack = Page.ClientScript.GetCallbackEventReference(this,
        "message", "processCallbackResult", "context", "postBackError", true);
}
```

接着要实现的是 ICallbackEventHandler 接口要求的两个方法，它们的实现代码如下：

```
void ICallbackEventHandler.RaiseCallbackEvent(string eventArgument) {
    callbackArg = eventArgument;
}

string ICallbackEventHandler.GetCallbackResult() {
```

```
    // 将传入参数依据&分割符分到 querystring 变量中
    Array keyValuePairs = callbackArg.Split("&".ToCharArray());
    NameValueCollection queryString = new NameValueCollection();
    string[] keyValue;
    string response = "";
    if (keyValuePairs.Length > 0) {
        for (int i = 0; i < keyValuePairs.Length; i++) {
            keyValue =
     keyValuePairs.GetValue(i).ToString().Split("=".ToCharArray());
            queryString.Add(keyValue[0], keyValue[1]);
        }
    }
    else {
        keyValue = callbackArg.Split("=".ToCharArray());
        if (keyValue.Length > 0)
            queryString.Add(keyValue[0], keyValue[1]);
    }


    // 针对参数中指定的 ActiveType 不同执行不同操作
    string controlType = queryString["ActiveType"];
    switch (controlType) {
        case "GetLegend": {

                // 根目录虚拟路径
                string virtualPath = Page.Request.ApplicationPath;
                // 根目录绝对路径
                string pathRooted =
HttpContext.Current.Server.MapPath(virtualPath);
                string savePath = pathRooted + "\\Output\\Legend.bmp";
                response = GisFunctionality.GetLegend(Map1, savePath);
                break;
            }
        default:
            break;
    }

    return response;
}
```

在 GetCallbackResult 方法中，我们利用 HttpServerUtility 类的 MapPath 方法来得到虚拟路径相对应的物理文件路径，然后加上 Output\Legend.bmp，这就是我们保存图例图片的文件所在绝对路径。因此需要读者在工程中创建一个 Output 文件夹。

实现保存图例的代码是 GisFunctionality 类的 GetLegend 类。

在工程中加入 ASP.NET 的文件夹 APP_CODE，并在其中加入 GisFunctionality 类。

首先在类中加入如下命名空间的引用：

```
using System.Drawing;
using ESRI.ArcGIS.ADF.Web;
using System.Collections.Generic;
using ESRI.ArcGIS.ADF.Web.DataSources;
```

```
using ESRI.ArcGIS.ADF.Web.UI.WebControls;
```

首先在 GetLegend 中加入如下代码：

```
public static string GetLegend(Map map, string savePath) {
    int printResolution = 96;
    int legendWidth = 200;
    int legendHeight = 200;
    int marginPadding = 10;
    int legendEntryPadding = 5;
    string legendTitle = "北美地图图例";
}
```

上述代码定义了绘制图例使用的参数，包括图片宽度、高度、分辨率以及名称。接着在方法中加入如下代码，得到图例信息：

```
Dictionary<string, KeyValuePair<string, CartoImage>> legendInfo = null;
legendInfo = new Dictionary<string, KeyValuePair<string, CartoImage>>();
int layerCounter = 0;
foreach (IMapResource item in map.MapResourceManagerInstance.GetResources())
{
    IMapTocFunctionality mtoc = null;
    mtoc = item.CreateFunctionality(typeof(IMapTocFunctionality), null)
        as IMapTocFunctionality;
    if (mtoc == null)
        continue;

    IMapFunctionality mf = null;
    foreach (IGISFunctionality entry in item.Functionalities) {
        if (entry is IMapFunctionality)
            mf = entry as IMapFunctionality;
    }
    if (mf == null || !mf.DisplaySettings.Visible)
        continue;

    TocDataFrame[] tocDataFrames = mtoc.GetMapContents(mf.Name,
        WebImageFormat.PNG24, true, false);
    foreach (TocDataFrame tocDataFrame in tocDataFrames) {
        foreach (TocLayer tocLayer in tocDataFrame) {
            if (tocLayer.Visible) {
                List<KeyValuePair<string, CartoImage>> legends = null;
                legends = GetLegendImages(tocLayer);
                if (legends != null && legends.Count > 0) {
                    foreach (KeyValuePair<string, CartoImage> entry in legends)
                    {
                        legendInfo.Add(layerCounter.ToString(),
                                      new KeyValuePair<string, CartoImage>
                                      (entry.Key, entry.Value));
                        layerCounter++;
                    }
                }

            }
```

```
        }
    }
}
```

在上面的代码中，对地图控件中的每个地图资源进行循环。由地图资源对象创建 Toc 功能，通过该功能的 GetMapContents 方法，得到 TocDataFrame 对象数组。然后对该数组进行循环。在该循环中对 TocDataFrame 对象中的 TocLayer 对象组进行循环。在该循环中以 TocLayer 对象为参数，调用 GetLegendImages 方法，得到该图层对应的 CartoImage 对象。该对象包含了制图需要的信息。

GetLegendImages 方法的代码如下：

```
private static List<KeyValuePair<string, CartoImage>>
GetLegendImages(TocLayer tocLayer)
{
    List<KeyValuePair<string, CartoImage>> legends =
new List<KeyValuePair<string, CartoImage>>();
    if (tocLayer != null && tocLayer.Visible) {
        if (tocLayer.TocSymbolGroupCount > 0) {
            if (tocLayer.TocSymbolGroupCount == 1) {
                TocSymbolGroup symbolGroupSingle =
tocLayer.GetTocSymbolGroup(0);

                if (symbolGroupSingle.Count == 1) {
                    TocSymbol tocSymbol = symbolGroupSingle[0];
                    legends.Add(new KeyValuePair<string, CartoImage>(
tocLayer.LayerName, tocSymbol.Image));
                }
                else {
                    System.Collections.IEnumerator e =
 tocLayer.GetTocSymbolGroups();
                    // 增加图层名称
                    if (e.MoveNext() && e.Current != null)
                        legends.Add(new KeyValuePair<string, CartoImage>(
tocLayer.LayerName, null));

                    e.Reset();
                    while (e.MoveNext()) {
                        // 增加组的名称
                        TocSymbolGroup symbolGroup = e.Current as TocSymbolGroup;
                        legends.Add(new KeyValuePair<string, CartoImage>(
symbolGroup.Heading, null));

                        foreach (TocSymbol tocSymbol in symbolGroup)
                        {
                            // 增加图例项组
                            legends.Add(new KeyValuePair<string, CartoImage>(
tocSymbol.Label, tocSymbol.Image));
                        }
                    }
                }
            }
            else {
```

```
                System.Collections.IEnumerator e =
    tocLayer.GetTocSymbolGroups();
                if (e.MoveNext() && e.Current != null)
                    legends.Add(new KeyValuePair<string,
    CartoImage>(tocLayer.LayerName, null));
                e.Reset();
                while (e.MoveNext()) {
                    TocSymbolGroup symbolGroup = e.Current as TocSymbolGroup;
                    legends.Add(new KeyValuePair<string,
 CartoImage>(symbolGroup.Heading, null));

                    foreach (TocSymbol tocSymbol in symbolGroup)
                    {
                        legends.Add(new KeyValuePair<string,
CartoImage>(tocSymbol.Label, tocSymbol.Image));
                    }
                }
            }
        }

        if (tocLayer.TocLayerCount > 0) {
            System.Collections.IEnumerator subLayers =
                            tocLayer.GetTocLayers();
            while (subLayers.MoveNext()) {
                List<KeyValuePair<string, CartoImage>> subLayerLegends =
 GetLegendImages(subLayers.Current as TocLayer);
                if (subLayerLegends != null)
                    legends.AddRange(subLayerLegends);
            }
        }
    }
    return legends;
}
```

该方法的目的就是得到一个 TocLayer 中所有的子 TocLayer。这是因为对于某些专题地图，图层的图例不止一个，例如独立值专题地图，每个要素都用一种不同的方式，所以该图层中有多少要素，那么就有多少个图例。

通过上述的代码，得到了所有图例的信息，包括子图例信息。回到 GetLegend 中，加入如下代码，该代码用于创建图例，并将其保存为一图片文件：

```
if (legendInfo == null || legendInfo.Count == 0)
    return "";
Bitmap legendImage = null;
legendImage = new Bitmap(legendWidth, legendHeight);
legendImage.SetResolution(printResolution, printResolution);
Graphics graphics = Graphics.FromImage(legendImage);
Font stringFont = new Font("Verdana", 8, System.Drawing.FontStyle.Bold);
SolidBrush drawBrush = new SolidBrush(Color.Black);
List<Bitmap> legendEntries = new List<Bitmap>();

foreach (KeyValuePair<string, KeyValuePair<string, CartoImage>>
```

```
    item in legendInfo) {
        if (item.Value.Value != null) {
            Bitmap swatch =
    new Bitmap(new System.IO.MemoryStream(
    item.Value.Value.MimeData.Bytes));
            float swatchColumnWidth =
    (swatch.Width * (printResolution / swatch.HorizontalResolution))
                > 100 ? 100 :
    (swatch.Width * (printResolution / swatch.HorizontalResolution));
            SizeF textSize = graphics.MeasureString(item.Value.Key, stringFont,
                (int)(legendWidth - swatchColumnWidth - (marginPadding * 2)));
            float inidividualHeight =
    (swatch.Height * (printResolution / swatch.VerticalResolution))
                 > textSize.Height ?
    (swatch.Height * (printResolution / swatch.VerticalResolution))
                 : textSize.Height;

            Bitmap individuallegend =
    new Bitmap(legendWidth, (int)inidividualHeight);
            individuallegend.SetResolution(printResolution, printResolution);
            Graphics g = Graphics.FromImage(individuallegend);
            g.DrawImage(swatch, marginPadding, 0);
            RectangleF textArea = new RectangleF(
    swatchColumnWidth + marginPadding, 0,
                legendWidth - swatchColumnWidth - (marginPadding * 2),
      inidividualHeight);
            g.DrawString(item.Value.Key, stringFont, drawBrush, textArea);
            g.Dispose();
            legendEntries.Add(individuallegend);
        }
        else {
            SizeF textSize = graphics.MeasureString(item.Value.Key, stringFont,
                (int)(legendWidth - (marginPadding * 2)));
            float inidividualHeight = textSize.Height;
            Bitmap individuallegend =
    new Bitmap(legendWidth, (int)inidividualHeight);
            individuallegend.SetResolution(printResolution, printResolution);
            Graphics g = Graphics.FromImage(individuallegend);
            RectangleF textArea = new RectangleF(marginPadding, 0,
                legendWidth - (marginPadding * 2), inidividualHeight);
            g.DrawString(item.Value.Key, stringFont, drawBrush, textArea);
            g.Dispose();
            legendEntries.Add(individuallegend);
        }
    }

    Font legendTitleFont = new Font("Verdana", 10, System.Drawing.FontStyle.Bold);
    SizeF legendTitleArea = graphics.MeasureString(legendTitle, legendTitleFont);

    float currentY = 0;
    //  在图例中加入图例名称
    graphics.DrawString(legendTitle, legendTitleFont,
```

```
                        drawBrush, marginPadding, legendEntryPadding);
currentY += legendTitleArea.Height + marginPadding;

//  循环加入每个图例项
foreach (Bitmap entry in legendEntries)
{
    if ((currentY + entry.Height) < legendHeight)
    {
        graphics.DrawImage(entry, 0, currentY);
        currentY += entry.Height + legendEntryPadding;
    }
}

legendImage.Save(savePath);
graphics.Dispose();
```

上述代码主要是利用.NET 中 GDI+绘制图例。最后在方法中加入如下代码，将 Image 控件的图片 src 属性指定为刚保存的图例文件：

```
string returnstring = savePath;
CallbackResult cr = new CallbackResult("image", "legend",
                                       "image", "Output\\Legend.bmp");
map.CallbackResults.Add(cr);
return map.CallbackResults.ToString();
```

编译并运行程序，可得到如图 5.21 所示的结果。

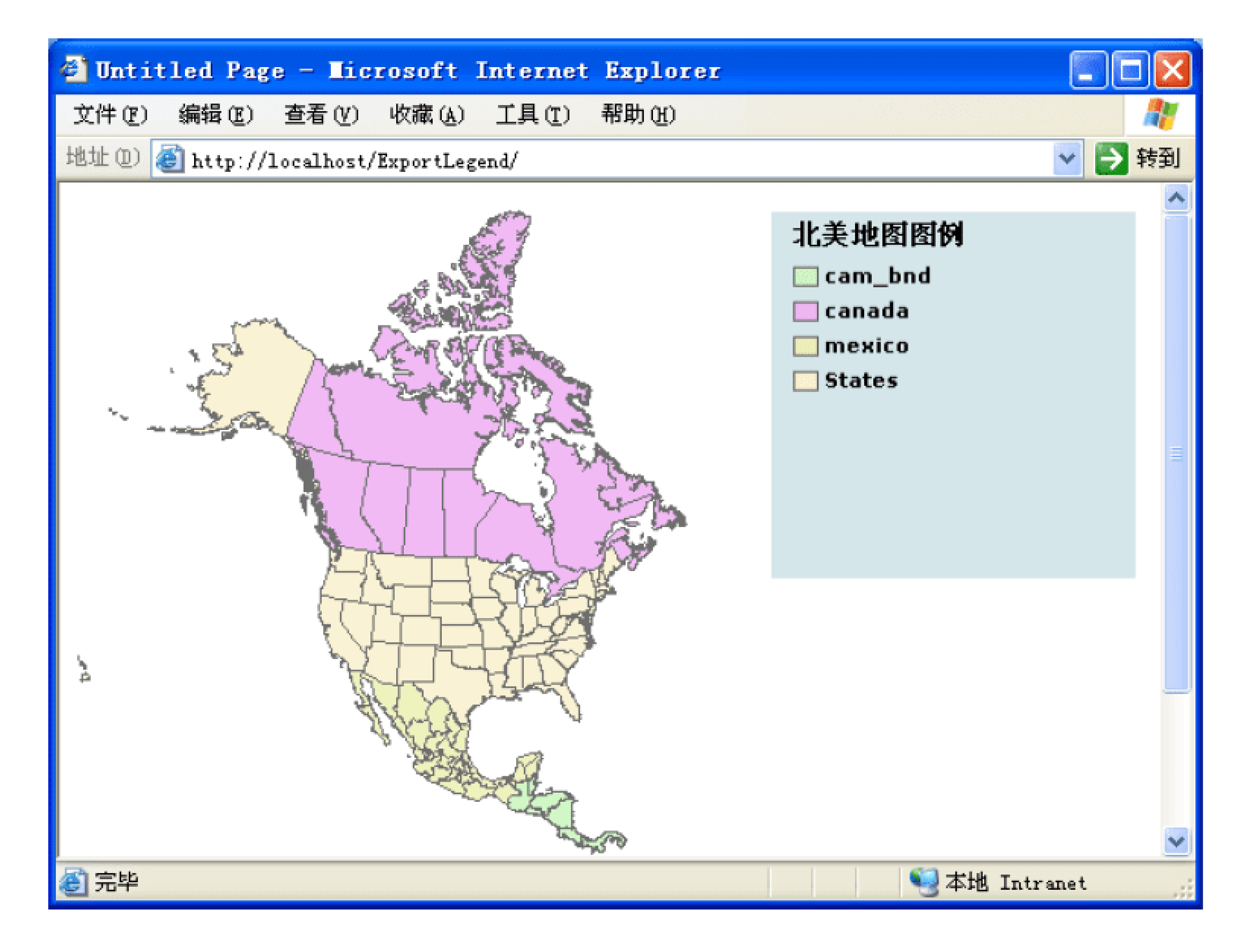
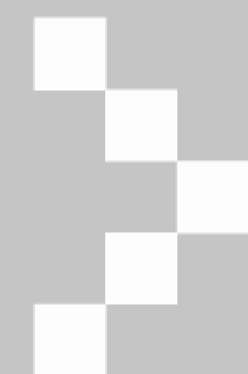


图 5.21　利用 Toc 功能动态创建图例

第 6 章



自定义数据源

在面向服务架构技术很成熟的今天，当开发一个应用系统时，特别是大一点的系统，用户一般有集成其他服务提供的数据的需求。例如在北京市开发地理信息系统，一般都要求集成北京市测绘设计研究院提供的基础行政区划图服务，以及北京市信息资源管理中心提供的遥感影像服务。对于这种需求，在 ArcGIS Server 9.2 的开发中，可以通过自定义数据源的方式来实现。

本章将通过两个实例来演示如何实现自定义数据源。通过本章你将了解到：

6.1 自定义数据源相关概念

正如前面多次提到的，Web ADF 通过在 ESRI.ARCGIS.ADF.Web.DataSources 中提供一组抽象接口，包括 IGISDataSource、IGISResource、IMapResource、IGISFunctionality 等，搭建了一个同时操作多种数据源的框架。并且 Web ADF 自身提供了 ArcGIS Server 数据源（在命名空间 ESRI.ARCGIS.ADF.Web.DataSources.ArcGISServer 中实现）、ArcIMS 数据源、ArcWeb 服务数据源等。在某种意义上来说，这些数据源也可以称为自定义数据源。

因此要自定义数据源，就必须实现 ESRI.ARCGIS.ADF.Web.DataSources 中一组接口，当然并不需要实现所有的接口，可以根据需要实现必要的接口即可。例如对于遥感影像数据就没有必要实现 IQueryFunctinality。再例如对于辅助数据源就没有必要实现 IScaleBarFunctionality 功能。

要自定义数据源，最重要的是要弄明白 ESRI.ARCGIS.ADF.Web.DataSources 中接口之间的关系。图 6.1 用 UML 的形式表达了自定义接口通常要实现的接口，以及这些接口之间的关系。

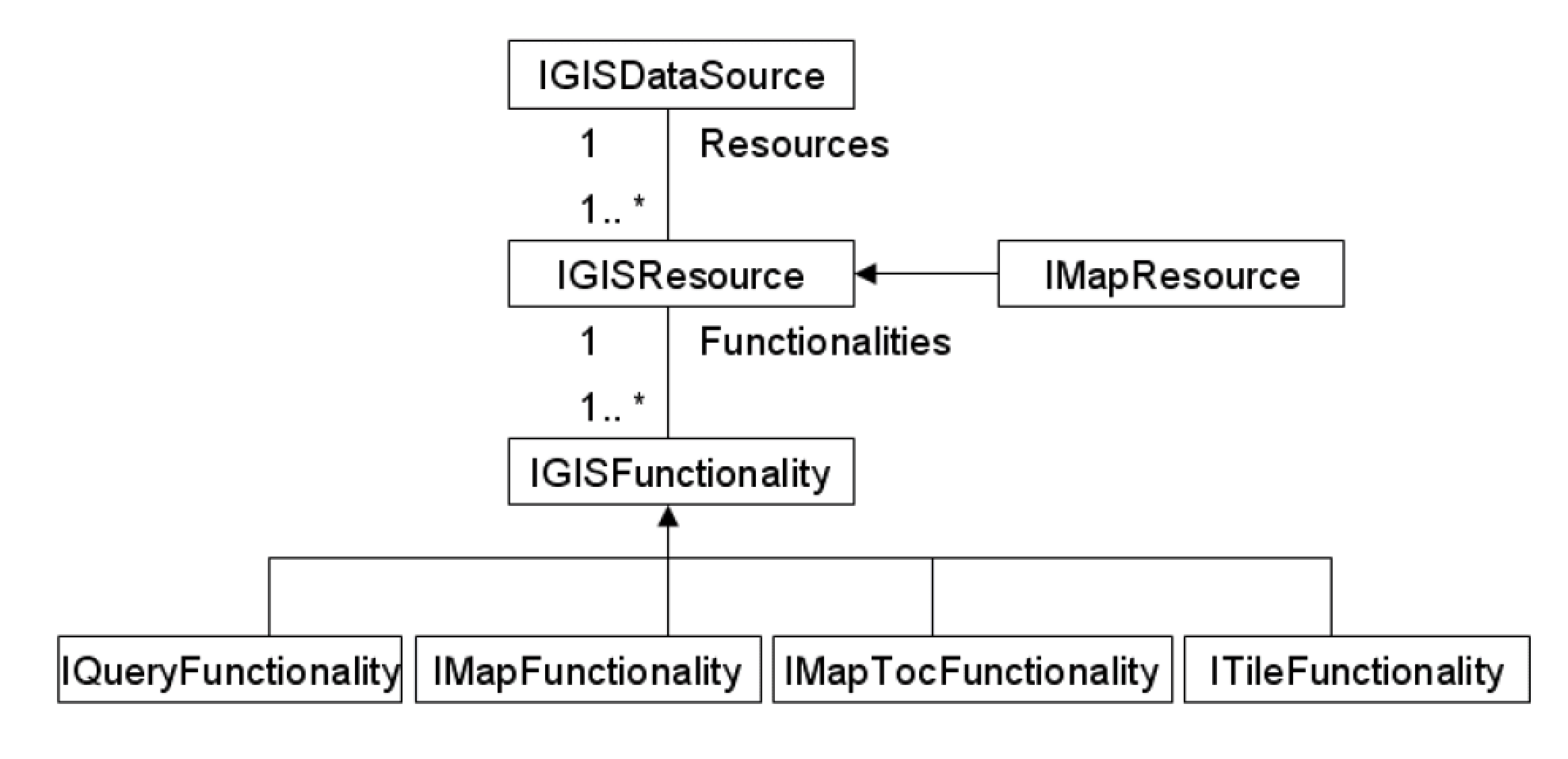


图 6.1 自定义接口通常要实现的接口之间的关系

对于自定义数据源，当然源头是代表数据源的 IGISDataSource，该接口中有一 Resources 属性，类型是 GISResourceCollection，表示一组资源的集合。也就是说数据源包含了一组 GIS 资源。其中地图资源是一种类型的 GIS 资源，也就是说 IMapResource 接口继承于 IGISResource。在 GIS 资源接口中则有一 Functionalities 属性，类型是 GISFunctionalityCollection，表示一组 GIS 功能的集合。也就是说 GIS 资源中包含一组 GIS 功能。GIS 功能的接口为 IGISFunctionality，其派生接口包括绘图功能 IMapFunctionality、查询功能 IQueryFunctionality、Toc 功能 IMapTocFunctionality 以及地图切片功能 ITileFunctionality 等。

因此对于自定义数据源，首先要实现的是 IGISDataSource 接口，另一个通常要实现的是 IMapResource 接口，实现该接口意味着也需要实现 IGISResource 接口，还有一般也要实现 IMapFunctionality 接口，由于该接口继承于 IGISFunctionality，因此也意味着要实现 IGISFunctionality 接口。

6.2　XML 数据源

6.2.1　数据格式

本实例要读入的第一种数据源是 XML 格式的空间数据，样本数据如下所示：

```
<?xml version="1.0" encoding="utf-8" ?>
<REXML version="1.0">
  <LAYER id="1" name="Trips">
    <SIMPLERENDERER>
      <SIMPLEMARKERSYMBOL color="0,255,0" type="star"
                          width="16" outlinecolor="0,0,0"/>
    </SIMPLERENDERER>
    <FEATURES>
      <FEATURE featureid="1">
        <FIELD name="SHAPE" type="-98" >
          <FIELDVALUE>
            <POINT x="-117.1" y="34.0" />
          </FIELDVALUE>
        </FIELD>
        <FIELD name="Status" type="12">
          <FIELDVALUE valuestring="Start Trip" />
        </FIELD>
      </FEATURE>
      <FEATURE featureid="2">
        <FIELD name="SHAPE" type="-98" >
          <FIELDVALUE>
            <POINT x="-120.6" y="39.5" />
          </FIELDVALUE>
        </FIELD>
        <FIELD name="Status" type="12">
          <FIELDVALUE valuestring="Mid Trip" />
        </FIELD>
      </FEATURE>
      <FEATURE featureid="3">
        <FIELD name="SHAPE" type="-98" >
          <FIELDVALUE>
            <POINT x="-110.5" y="34.7" />
          </FIELDVALUE>
        </FIELD>
        <FIELD name="Status" type="12">
          <FIELDVALUE valuestring="End Trip" />
        </FIELD>
      </FEATURE>
      <FEATURE featureid="4">
        <FIELD name="SHAPE" type="-98" >
          <FIELDVALUE>
```

```
            <POINT x="-120.5" y="44.7" />
          </FIELDVALUE>
        </FIELD>
        <FIELD name="Status" type="12">
          <FIELDVALUE valuestring="End Trip" />
        </FIELD>
      </FEATURE>
    </FEATURES>
  </LAYER>
</REXML>
```

文件结构很简单，每个图层数据包含在<LAYER>与</LAYER >标签之间，其中SIMPLERENDERER定义了着色器。每个要素定义在<FEATURE>与</FEATURE>之间。要素中的子项是字段（FIELD），名称为SHAPE的字段表示为空间数据，其他字段表示属性数据。

6.2.2 实现数据源接口

在 Visual Studio 2005 中选择 File 菜单的 New Project 命令，打开如图 6.2 所示的对话框。

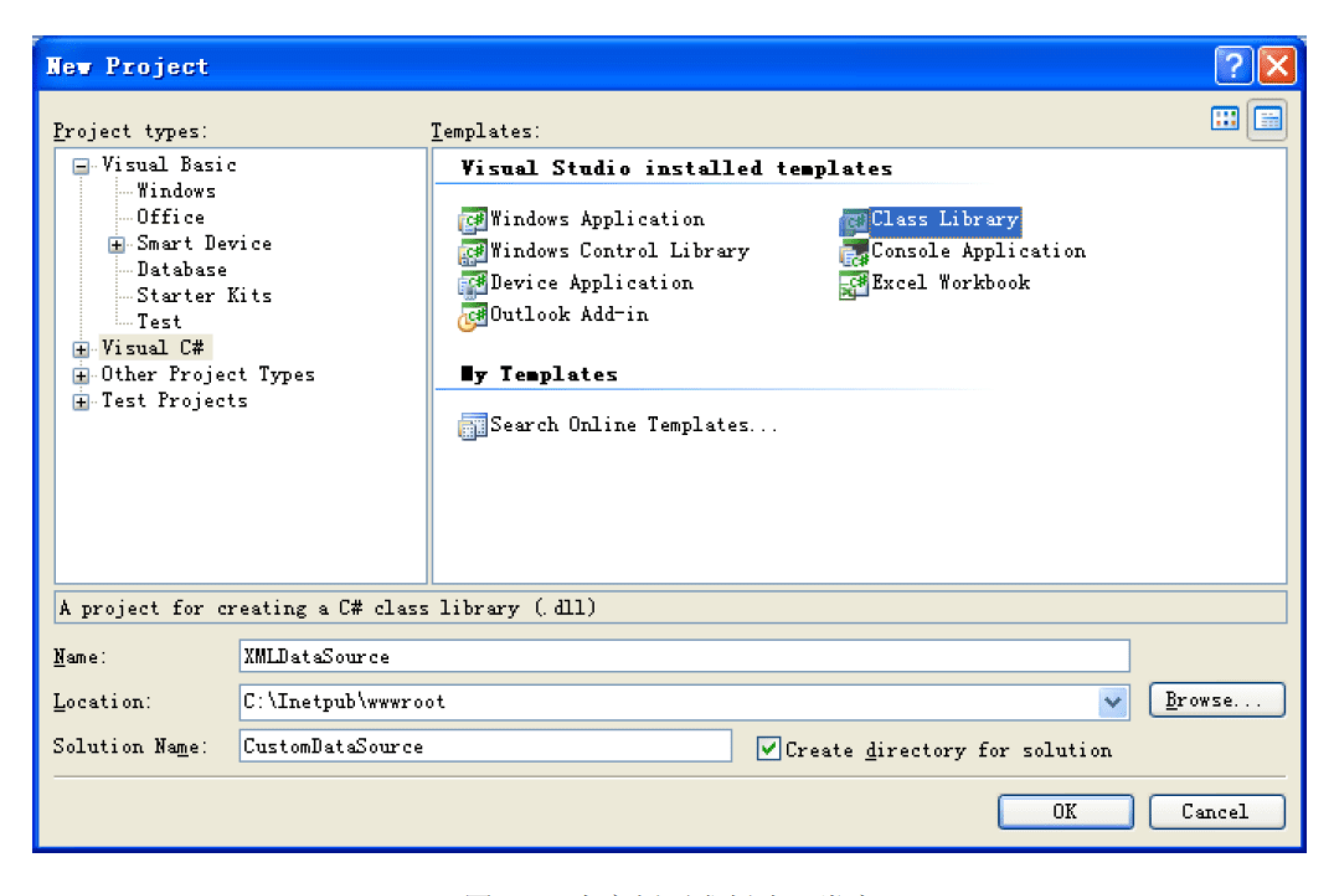


图 6.2 本实例要求创建一类库

为了能在多个应用程序中使用，因此模板选择创建 Class Library，即类库，将工程名称设置为 XMLDataSource，将解决方案名称修改为 CustomDataSource（默认与工程同名）。我们将在该解决方案中加入其他工程。

在解决方案管理面板中，将默认加入的 Class1.cs 文件名修改为 GISDataSource.cs，编辑环境会自动将类名由 Class1 改变为 GISDataSource。我们将利用该类来实现 IGISDataSource 接口。

为了能利用 Web ADF 中的类库以及.NET 中的类库，先通过工程右键菜单的 Add Reference 命令，加入 System.Data、System.Web、ESRI.ArcGIS.ADF、ESRI.ArcGIS.ADF.Web、ESRI.ArcGIS.ADF.Web.DataSource 以及 ESRI.ArcGIS.ADF.Web.UI.WebControls 类库的引用。

IGISDataSource 接口需要实现的属性与方法如图 6.3 所示。有了接口图，对于实现该接口就容易多了。

IGISDataSource
DataSourceDefinition: System.String Identity: System.String Initialized: System.Boolean Name: System.String Page: System.Web.UI.Page Resources: GISResourceCollection State: System.Collections.Hashtable
CreateResource (System.String resourceDefinition, System.String name): IGISResource Dispose: System.Void GetAvailableResourceDefinitions (System.Type resourceType): System.String[] Initialize: System.Void LoadState (System.Collections.Hashtable state): System.Void SaveState: System.Collections.Hashtable

图 6.3　IGISDataSource 中的接口与方法

首先在文件的头部加入如下命名空间的引用：

```
using System.Collections;
using System.Web.UI;
using ESRI.ArcGIS.ADF.Web.DataSources;
```

然后将类的声明代码修改为如下代码，声明实现 IGISDataSource 接口：

```
public class GISDataSource : IGISDataSource
```

下面需要做的就是根据接口图来实现接口。对于属性比较好实现，先定义对应的 private 类型的字段，命名也保持一致，只是将字段名的第一个字母改为小写。

在 GISDataSource 类中加入接口规定的属性对应的字段，代码如下（每个字段的意义见对应属性的注释）：

```
private string dataSourceDefinition = string.Empty;
private string identity = string.Empty;
bool initialized = false;
private string name = string.Empty;
private Page page = null;
private GISResourceCollection resources = new GISResourceCollection();
private Hashtable state;
```

然后加入如下三个构造方法：

```
    public GISDataSource() {
    }

    public GISDataSource(string name, string dataSourceDefinition)
        : this(name, string.Empty, dataSourceDefinition) {
    }

    public GISDataSource(string name, string identity,
    string dataSourceDefinition) {
        this.name = name;
        this.identity = identity;
        this.dataSourceDefinition = dataSourceDefinition;
    }
```

接着要实现的就是 IGISDataSource 接口了，首先来实现其属性。属性的实现很简单，对于 get 操作，返回对应字段，对于 set，将对应只段设置为 value 即可。要注意的是有些属性是只读的，因此没有 set，而有些属性是只写的，没有 get。属性的实现代码如下：

```
    /// <summary>
    /// GIS 数据源的名称
    /// </summary>
    public string Name {
        get {
            return name;
        }
        set {
            name = value;
        }
    }

    /// <summary>
    /// 数据源的定义信息，通常存储的数据源的位置
    /// </summary>
    public string DataSourceDefinition {
        get {
            return dataSourceDefinition;
        }
        set {
            if (dataSourceDefinition != value) {
                dataSourceDefinition = value;
            }
        }
    }

    /// <summary>
    /// 连接数据源的用户身份
    /// </summary>
    public string Identity {
        get {
            return identity;
```

```
        }
        set {
            identity = value;
        }
    }

    /// <summary>
    /// 管理该数据源对象的页面控件
    /// </summary>
    public Page Page {
        get {
            return page;
        }
        set {
            page = value;
        }
    }

    /// <summary>
    /// 该数据源能够支持的资源集合
    /// </summary>
    public GISResourceCollection Resources {
        get {
            return resources;
        }
        set {
            resources = value;
        }
    }

    /// <summary>
    /// 数据源是否被初始化
    /// </summary>
    public bool Initialized {
        get {
            return initialized;
        }
    }

    /// <summary>
    /// 该数据源状态的哈希表。与数据源连接的资源与功能可能将它们的状态存储在该属性中。
DataSourceManager 控件存储该哈希表
    /// </summary>
    public Hashtable State {
        get {
            return state;
        }
    }
```

对于数据源接口的方法实现也很简单，代码如下：

```
/// <summary>
/// 从 DataSourceManager 控件维护的哈希表中加载状态
/// </summary>
/// <param name="state">存储状态的哈希表</param>
public void LoadState(Hashtable state) {
    this.state = state;
}

/// <summary>
/// 初始化数据源
/// </summary>
public void Initialize() {
    initialized = true;
}
/// <summary>
/// 保存状态
/// </summary>
/// <returns>存储状态的哈希表</param>
public Hashtable SaveState() {
    return state;
}
/// <summary>
/// 释放数据源
/// </summary>
public void Dispose() {
    initialized = false;
}

/// <summary>
/// 返回使用该数据源的资源的定义字符串
/// </summary>
public string[] GetAvailableResourceDefinitions(System.Type resourceType) {
    int i = 0;
    string[] definitions = new string[resources.Count];
    foreach (IGISResource rs in resources) {
        definitions[i] = rs.ResourceDefinition;
    }
    return definitions;
}
/// <summary>
/// 根据参数的定义创建资源
/// </summary>
/// <param name="resourceDefinition">要创建的资源的定义</param>
/// <param name="name">要创建的资源的名称</param>
/// <returns>参数指定定义的 GIS 资源</returns>
public IGISResource CreateResource(string resourceDefinition, string name)
{
    throw new Exception("The method or operation is not implemented.");
}
```

正如如上代码所示，对于有些方法，只要不调用，可以不编写实现代码。

这就完成了数据源接口的实现。第二个要实现的接口是 IMapResource 接口。

6.2.3　实现地图资源接口

地图资源接口负责与数据源连接，从数据源得到数据，并创建与维护功能。

由于 IMapResource 接口又继承了 IGISResource 接口，所以要实现地图资源接口，必须同时实现上述两接口。两个接口包含的属性与方法如图 6.4 所示。

IGISResource

DataSource: IGISDataSource
Functionalities: GISFunctionalityCollection
Initialized: System.Boolean
Name: System.String
ResourceDefinition: System.String
ValidationTimeout: System.Int32

ClearState: System.Void
CreateFunctionality (System.Type functionalityType, System.String functionalityName): IGISFunctionality
Dispose: System.Void
Initialize: System.Void
LoadState: System.Void
SaveState: System.Void
SupportsFunctionality (System.Type functionalityType): System.Boolean

IMapResource

DisplaySettings: ESRI.ArcGIS.ADF.Web.DisplaySettings
MapInformation: IMapInformation

图 6.4　IMapResource 与 IGISResource 接口的属性与方法

在工程增加一个类，命名为 MapResource。

在该类声明代码之前加入如下命名空间的引用：

```
using ESRI.ArcGIS.ADF.Web.Display.Graphics;
using ESRI.ArcGIS.ADF.Web.Display.Drawing;
using ESRI.ArcGIS.ADF.Web;
using ESRI.ArcGIS.ADF.Web.Geometry;
using ESRI.ArcGIS.ADF.Web.SpatialReference;
using ESRI.ArcGIS.ADF.Web.DataSources;
using ESRI.ArcGIS.ADF.Web.Display.Symbol;
using System.Collections;
using System.Xml;
```

将 MapResource 类的声明修改为如下代码，增加对地图资源接口实现的声明：

```
public class MapResource : IMapResource
```

然后利用开发环境的代码自动功能，创建 IMapResource 与 IGISResource 接口要实现的属性与方法的框架。通过将光标移动到 IMapResource 字符串上，在第一个字母下出现一下划线时，将鼠

标移动到该下划线上，将出现如图 6.5 所示的界面，在其中选择任何一项都可实现自动生成代码框架。

```
namespace XMLDataSource
{
    public class MapResource : IMapResource
    {
        public MapResource()
        {
        }

    }
}
```

```
interface ESRI.ArcGIS.ADF.Web.DataSources.IMapResource
Implement interface 'IMapResource'
Explicitly implement interface 'IMapResource'
```

图 6.5　利用集成开发环境代码生成功能自动加入接口需要实现的属性与方法

由于有两个接口需要实现，因此自动生成的代码利用 region 语句将它们区分开了。我们可以利用该功能，将各自要求实现的属性的对应字段放在各个代码段的开头，在地图资源接口实现代码段，加入如下两个字段，及对应属性代码：

```
#region IMapResource Members

private IMapInformation mapInformation = null;
private DisplaySettings displaySettings = null;

/// <summary>
/// 地图信息，例如数据本身的空间参考与范围等
/// </summary>
public IMapInformation MapInformation {
    get {
        return mapInformation;
    }
    set {
        mapInformation = value;
    }
}

/// <summary>
/// 地图显示信息，例如背景颜色、透明度等
/// </summary>
public DisplaySettings DisplaySettings {
    get {
        return displaySettings;
    }
    set {
        displaySettings = value;
    }
}

#endregion
```

在 GIS 资源接口代码段加入该接口规定要实现的属性，及其对应字段：

```
    private IGISDataSource dataSource = null;
    private     GISFunctionalityCollection     functionalities     =     new
GISFunctionalityCollection();
    bool initialized = false;
    private string name = string.Empty;
    private string resourceDefinition = string.Empty;
    private int validationtimeout = 0;

    /// <summary>
    /// 判断资源是否有效的时间，以毫秒为单位
    /// </summary>
    public int ValidationTimeout
    {
        get
        {
            return validationtimeout;
        }
        set
        {
            validationtimeout = value;
        }
    }

    /// <summary>
    /// GIS 资源的名称
    /// </summary>
    public string Name {
        get {
            return name;
        }
        set {
            name = value;
        }
    }

    /// <summary>
    /// 该资源的定义
    /// </summary>
    public string ResourceDefinition {
        get {
            return resourceDefinition;
        }
        set {
            resourceDefinition = value;
        }
    }

    /// <summary>
    /// 该资源归属的数据源
    /// </summary>
```

```
public IGISDataSource DataSource {
    get {
        return dataSource;
    }
    set {
        dataSource = value;
    }
}

/// <summary>
/// 该资源能支持的功能集合
/// </summary>
public GISFunctionalityCollection Functionalities {
    get {
        return functionalities;
    }
    set {
        functionalities = value;
    }
}
```

在 MapResource 类自身代码段部分，加入如下代码：

```
public MapResource() {
}

public MapResource(string name, GISDataSource dataSource) {
    this.name = name;
    this.dataSource = dataSource;
}

private GraphicsDataSet graphics = null;
public GraphicsDataSet Graphics {
    get {
        return graphics;
    }
}

private string key {
    get {
        string tmp = this.GetType().ToString() + ":" + name;
        return tmp;
    }
}
```

在上述代码中，加入了一个图形数据集类型的字段 graphics，我们用该字段存放数据源对应的数据。

下面要实现的是 GIS 资源接口的方法。该接口最先执行的是资源初始化 Initialize 方法，在该方法中，需要从数据源中读入数据。该方法的代码如下：

```
public void Initialize() {
```

```
        if (mapInformation == null) {
            graphics = new GraphicsDataSet();
            string dataSourceConfig = DataSource.DataSourceDefinition;
            ProcessXML(dataSourceConfig);

            mapInformation = new MapInformation(graphics);
            if (DisplaySettings != null)
                graphics.ImageDescriptor = displaySettings.ImageDescriptor;
        }
        initialized = true;
    }
```

在上面的代码中，首先利用 ProcessXML 方法从数据源对象的 DataSourceDefinition 属性指定的文件中读入数据。然后根据图形数据，创建一个地图信息对象。该 MapInformation 是本实例在后面要实现的。

ProcessXML 方法的代码如下：

```
    private void ProcessXML(string dataSourceConfig) {
        try {
            // 从文件中读入 XML 数据
            XmlDocument doc = new XmlDocument();
            doc.Load(dataSourceConfig);
            XmlNode root = doc.DocumentElement;

            // 得到图层信息
            XmlNode layerNode = root.SelectSingleNode("LAYER");
            string layername = layerNode.Attributes.GetNamedItem("name").Value;
            string layerid = layerNode.Attributes.GetNamedItem("id").Value;

            // 创建图形图层用于存储要素
            ESRI.ArcGIS.ADF.Web.Display.Graphics.ElementGraphicsLayer  glayer  =
null;
            glayer =
                new ESRI.ArcGIS.ADF.Web.Display.Graphics.ElementGraphicsLayer();
            glayer.TableName = layername;

            // 读着色器信息
            XmlNode rendererNode = layerNode.SelectSingleNode("SIMPLERENDERER");
            XmlNode markerNode =
                rendererNode.SelectSingleNode("SIMPLEMARKERSYMBOL");
            string markercolor =
                markerNode.Attributes.GetNamedItem("color").Value;
            string markertype = markerNode.Attributes.GetNamedItem("type").Value;
            string markerwidth =
                markerNode.Attributes.GetNamedItem("width").Value;
            string markeroutlinecolor =
                markerNode.Attributes.GetNamedItem("outlinecolor").Value;

            // 创建符号
            SimpleMarkerSymbol sms = new SimpleMarkerSymbol();
```

```
string[] markerrgb = markercolor.Split(',');
int markerr = Int32.Parse(markerrgb[0]);
int markerg = Int32.Parse(markerrgb[1]);
int markerb = Int32.Parse(markerrgb[2]);
sms.Color = System.Drawing.Color.FromArgb(markerr, markerg, markerb);
switch (markertype) {
    case "circle":
        sms.Type = MarkerSymbolType.Circle;
        break;
    case "square":
        sms.Type = MarkerSymbolType.Square;
        break;
    case "triangle":
        sms.Type = MarkerSymbolType.Triangle;
        break;
    case "star":
        sms.Type = MarkerSymbolType.Star;
        break;
    default:
        break;
}
int markerwidthInt;
if (!Int32.TryParse(markerwidth, out markerwidthInt)) {
    markerwidthInt = 10;
}
sms.Width = markerwidthInt;
string[] markeroutrgb = markeroutlinecolor.Split(',');
int markeroutr = Int32.Parse(markeroutrgb[0]);
int markeroutg = Int32.Parse(markeroutrgb[1]);
int markeroutb = Int32.Parse(markeroutrgb[2]);
sms.OutlineColor =
    System.Drawing.Color.FromArgb(markeroutr, markeroutg, markeroutb);

// 读要素信息
XmlNode featuresNode = layerNode.SelectSingleNode("FEATURES");
XmlNodeList featureNodes = featuresNode.SelectNodes("FEATURE");

// 读图形信息，并创建图形对象
for (int t = 0; t < featureNodes.Count; ++t) {
    XmlNodeList flds = featureNodes[t].SelectNodes("FIELD");
    for (int s = 0; s < flds.Count; s++) {
        if (flds[s].Attributes["name"].Value == "SHAPE") {
            XmlNodeList fldvalues =
               flds[s].SelectSingleNode("FIELDVALUE").ChildNodes;
            if (fldvalues.Count < 1) {
                break;
            }
            else if (fldvalues.Count == 1) {
```

```
                        XmlNode fldvalue = fldvalues[0];
                        if (fldvalue.Name == "POINT") {
                            double dx =
                              Double.Parse(fldvalue.Attributes["x"].Value);
                            double dy =
                              Double.Parse(fldvalue.Attributes["y"].Value);

                            Point point = new Point(dx, dy);
                            GraphicElement ge = new GraphicElement(point, sms);
                            glayer.Add(ge);
                        }
                    }
                    else if (fldvalues.Count > 1) {}
                }
            }
        }

        graphics.Tables.Add(glayer);
    }
    catch (Exception ex) {
        throw new Exception("异常：不能加载 XML 数据" + ex.Message);
    }
}
```

上述方法就是利用处理 XML 文档的类，比如 XmlDocument、XmlNode 等，解析 6.2.1 节中给出的文件内容，并根据图形信息创建图形对象。

GIS 资源接口中另两个重要方法是 SupportsFunctionality 与 CreateFunctionality。前者用于判断资源是否支持某种类型的功能。该方法的代码如下：

```
public bool SupportsFunctionality(System.Type functionalityType) {
    if (functionalityType == typeof(IMapFunctionality))
        return true;
    else if (functionalityType == typeof(IMapTocFunctionality))
        return true;
    else
        return false;
}
```

上述代码表示，本资源只支持绘图功能与 Toc 功能。

CreateFunctionality 方法用于创建某种类型的功能，由于本实例的资源只支持绘图功能与 Toc 功能，因此针对该两种类型，分别利用 new 来创建两个功能类的实例。这两个功能类就是我们在随后章节要实现的。该方法的代码如下：

```
public IGISFunctionality CreateFunctionality(System.Type functionalityType,
string functionalityName)
{
    IGISFunctionality func = null;
    if (functionalityType == typeof(IMapFunctionality)) {
        func = new MapFunctionality(functionalityName, this);
    }
```

```
    else if (functionalityType == typeof(IMapTocFunctionality)) {
        func = new MapTocFunctionality(functionalityName, this);
    }
    else {
        throw new ArgumentException("functionalityType");
    }
    return func;
}
```

GIS 资源接口中另一大类的方法是用于状态的管理。LoadState 方法用于从数据源存储的状态中加载数据，代码如下：

```
public void LoadState() {
    if (dataSource == null)
        return;
    if (dataSource.State == null)
        return;

    object o = dataSource.State[key];
    if (o != null) {
        MapResource mr = o as MapResource;
        graphics = mr.graphics;
        mapInformation = mr.mapInformation;
        displaySettings = mr.displaySettings;
    }
}
```

而 SaveState 用于在数据源中保存当前资源的状态。代码如下：

```
public void SaveState() {
    if (dataSource == null)
        return;
    if (dataSource.State == null)
        return;
    dataSource.State[key] = this;
}
```

ClearState 方法用于清除资源的状态，代码如下：

```
public void ClearState() {
    this.graphics = null;
    this.mapInformation = null;
    this.displaySettings = null;
}
```

最后一个方法是 Dispose，用于销毁该资源，对于我们，只需要简单地将初始化标志设置为 false 即可，代码如下：

```
public void Dispose() {
    initialized = false;
}
```

至此就完成了地图资源的实现。下一步要实现的是绘图功能。

6.2.4　实现绘图功能

顾名思义，绘图功能要实现的主要就是生成一张地图图片。

由于 IMapFunctionality 接口又继承了 IGISFunctionaltiy 接口，所以对于要实现绘图功能接口，也必须同时实现上述两接口。两个接口包含的属性与方法如图 6.5 所示。

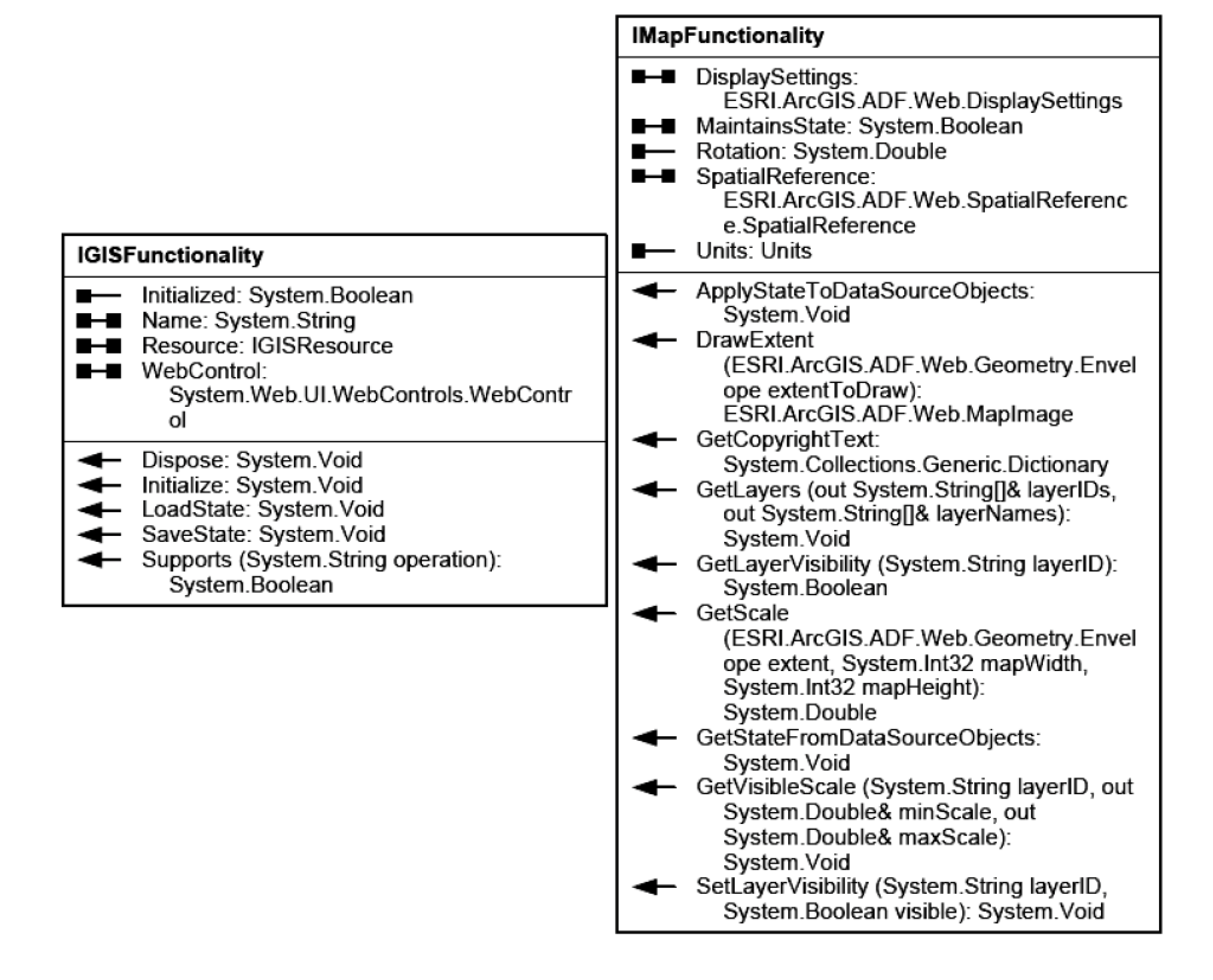


图 6.5　IMapFunctionality 与 IGISFunctionaltiy 接口的属性与方法

在工程中新加一个类，命名为 MapFunctionality。在其中先加入如下命名空间的引用：

```
using System.Collections;
using ESRI.ArcGIS.ADF.Web.Geometry;
using ESRI.ArcGIS.ADF.Web.SpatialReference;
using ESRI.ArcGIS.ADF.Web.DataSources;
using ESRI.ArcGIS.ADF.Web.Display.Graphics;
using ESRI.ArcGIS.ADF.Web.Display.Drawing;
using ESRI.ArcGIS.ADF.Web;
```

将 MapFunctionality 类的声明修改为如下代码，增加对绘图功能接口实现的声明：

```
class MapFunctionality : IMapFunctionality
```

利用集成开发环境的自动生成代码功能，生成 IMapFunctionality 接口代码框架。接着在绘图功能接口代码段，加入其要求实现的属性，代码如下：

```
private DisplaySettings displaySettings = null;
private bool maintainsState = false;
private SpatialReference spatialReference = null;

public bool MaintainsState {
    get {
        return maintainsState;
    }
    set {
        maintainsState = value;
    }
}

public DisplaySettings DisplaySettings {
    get {
        if (maintainsState) {
            if (displaySettings == null) {
                displaySettings = MapResource.DisplaySettings.Clone()
                        as DisplaySettings;
            }
            return displaySettings;
        }
        else
            return MapResource.DisplaySettings;
    }
    set {
        if (maintainsState)
            displaySettings = value;
        else
            MapResource.DisplaySettings = value;
    }
}

public ESRI.ArcGIS.ADF.Web.DataSources.Units Units {
    get  {
        throw new NotImplementedException();
    }
}
```

接着在 IGISFunctionality 接口代码段，加入其要求的属性的实现代码，代码如下：

```
bool initialized = false;
private string name = string.Empty;
private IGISResource resource = null;
System.Web.UI.WebControls.WebControl webControl;

public string Name {
    get {
        return name;
    }
    set {
```

```
            name = value;
        }
    }

    public IGISResource Resource {
        get {
            return resource;
        }
        set {
            resource = value;
        }
    }

    public bool Initialized {
        get {
            return initialized;
        }
    }

    public System.Web.UI.WebControls.WebControl WebControl {
        get {
            return webControl;
        }
        set {
            webControl = value;
        }
    }
```

接着在 MapFunctionaltiy 类代码段，加入如下代码：

```
    public MapFunctionality(string name, MapResource resource) {
        this.name = name;
        this.resource = resource;
    }

    public MapResource MapResource {
        get {
            return resource as MapResource;
        }
    }

    private GraphicsDataSet graphics = null;
    public GraphicsDataSet GraphicsDataSet {
        get {
            if (!maintainsState) {
                MapResource mr = MapResource;
                return mr == null ? null : mr.Graphics;
            }
            else {
                if (graphics != null) {
                    return graphics;
```

```
                }
                else {
                    MapResource mr = MapResource;
                    if (mr != null && mr.Graphics != null) {
                        graphics = mr.Graphics.Clone();
                    }

                    return graphics;
                }
            }
        }
    }

    private string key {
        get {
            string szResource = resource.GetType().ToString() + ":" + resource.Name;
            string szThis = this.GetType().ToString() + ":" + name;
            return (szResource + "," + szThis);
        }
    }
```

在上面的代码段中，利用 graphicsDataSet 字段存储数据源的图形数据，在其对应的属性代码中，首先查询功能是否保存状态，如果不保存状态，则从对应的地图资源中获取图形数据，如果保存状态，但是图形数据不存在时，从地图资源中拷贝图形数据。

绘图功能接口中 GetLayer Visibility 方法和 Set Louyer Visibility 方法用于控制图层的可见性，其中 GetLayerVisibility 方法用于得到某个图层的可见性，代码如下：

```
    public bool GetLayerVisibility(string layerID) {
        return getLayer(layerID).Visible;
    }

    private GraphicsLayer getLayer(string layerID) {
        GraphicsLayer   graphicslayer   =   GraphicsDataSet.Tables[layerID]   as
GraphicsLayer;
        return graphicslayer;
    }
```

SetLayerVisibility 方法用于设置图层的可见性，实现代码如下：

```
    public void SetLayerVisibility(string layerID, bool visible) {
        getLayer(layerID).Visible = visible;
    }
```

DrawExtent 当然是绘图功能接口中最重要的方法了，该方法用于绘制地图。在本实例中，直接调用 GraphicsDataSet 类的 DrawExtent 方法即可实现绘制地图。代码如下：

```
    public ESRI.ArcGIS.ADF.Web.MapImage DrawExtent
        (ESRI.ArcGIS.ADF.Web.Geometry.Envelope extentToDraw) {
        if (GraphicsDataSet != null) {
            ApplyStateToDataSourceObjects();
            return GraphicsDataSet.DrawExtent(extentToDraw);
```

```
        }
        else
            return null;
    }
```

在绘图功能接口中，还有一类方法用于获取与应用状态，其中 ApplyStateToDataSourceObjects 方法用于将更新的状态应用于地图资源。本实例实现代码如下：

```
    public void ApplyStateToDataSourceObjects() {
        if (GraphicsDataSet != null) {
            if (DisplaySettings != null)
                GraphicsDataSet.ImageDescriptor = DisplaySettings.ImageDescriptor;
        }
    }
```

GetStateFromDataSourceObjects 过程正好相反，从地图资源中得到状态信息。实现代码如下：

```
    public void GetStateFromDataSourceObjects() {
        if (GraphicsDataSet != null) {
            if (DisplaySettings != null)
                DisplaySettings.ImageDescriptor = GraphicsDataSet.ImageDescriptor;
        }
    }
```

对于本实例，不需要利用绘图功能接口中的其他方法，因此可直接保留集成开发环境生成的代码，而不需要修改。

下面要实现的是 GIS 功能接口的方法。GIS 功能接口中一类方法用于维护状态，其中方法 LoadState 用于根据不同设置从不同对象中加载状态信息，详细信息见实现如下代码：

```
    public void LoadState() {
        if (resource == null || resource.DataSource == null
            || resource.DataSource.State == null) {
            throw new Exception("资源无效");
        }

        // 如果状态保存过，则加载以前的状态
        object o = resource.DataSource.State[key];
        if (o != null) {
            MapFunctionality mf = o as MapFunctionality;
            // 从保存的状态中得到属性
            this.maintainsState = mf.MaintainsState;
            this.webControl = mf.WebControl;
            this.spatialReference = mf.spatialReference;
            // 如果维护状态，则加载本功能自身保存的状态
            if (maintainsState) {
                displaySettings = mf.displaySettings;
                graphics = mf.graphics;
            }
        }

        // 从数据源对象中得到共享属性，这些属性将依据数据源对象初始化
```

```
        GetStateFromDataSourceObjects();
    }
```

SaveState 方法用于在资源中保存状态。实现代码如下：

```
    public void SaveState() {
        if (resource == null)
            return;
        if (resource.DataSource == null)
            return;
        if (resource.DataSource.State == null)
            return;
        ApplyStateToDataSourceObjects();
        resource.DataSource.State[key] = this;
    }
```

GIS 功能接口中另一类方法用于初始化与销毁对象，分别对应 Initialize 与 Dispose 方法。实现代码如下：

```
    public void Initialize() {
        initialized = true;
    }

    public void Dispose() {
        initialized = false;
    }
```

接口中最后一个方法是 Supports，该方法用于判断本功能对象是否支持某类操作。由于本实例不准备实现计算比例尺功能，因此对于该操作，返回 false，表示不支持。对于其他操作，则返回 true。实现代码如下：

```
    public bool Supports(string operation) {
        if (operation == "GetScale")
            return false;
        return true;
    }
```

至此实现了绘图功能。下面要实现的是 Toc 功能。

6.2.5 实现 Toc 功能

Toc 功能就是要支持在 Toc 控件中操纵对应的地图资源，例如设置图层的可见性。

同绘图功能一样，IMapTocFunctionality 接口继承与 IGISFunctionality 接口，因此要实现 IMapTocFunctionality 接口，也必须同时实现 IGISFunctionality 接口。IMapTocFunctionality 接口要求实现的属性与方法如图 6.6 所示。

IMapTocFunctionality
← GetMapContents (System.String mapFunctionalityName, ESRI.ArcGIS.ADF.Web.WebImageFormat format, System.Boolean useMimeData, System.Boolean showAllDataFrames): ESRI.ArcGIS.ADF.Web.TocDataFrame[]

图 6.6　IMapTocFunctionality 接口的属性与方法

在工程中新加一个类，命名为 MapTocFunctionality。在其中先加入如下命名空间的引用：

```
using System.Collections;
using ESRI.ArcGIS.ADF.Web.Geometry;
using ESRI.ArcGIS.ADF.Web.SpatialReference;
using ESRI.ArcGIS.ADF.Web.DataSources;
using ESRI.ArcGIS.ADF.Web.Display.Graphics;
using ESRI.ArcGIS.ADF.Web.Display.Drawing;
using ESRI.ArcGIS.ADF.Web;
```

将 MapTocFunctionality 类的声明修改为如下代码，增加对 Toc 功能接口实现的声明：

```
class MapTocFunctionality: IMapTocFunctionality
```

利用集成开发环境的自动生成代码功能，生成 IMapTocFunctionality 接口代码框架。由于 Toc 功能接口中没有定义属性，因此直接在 GIS 功能接口代码段，加入其要求实现的属性与方法，代码如下：

```
#region IGISFunctionality Members
private string name = string.Empty;
private IGISResource resource = null;
private bool initialized = false;
private MapResource mapResource;
System.Web.UI.WebControls.WebControl webControl;
public System.Web.UI.WebControls.WebControl WebControl {
    get {
        return webControl;
    }
    set {
        webControl = value;
    }
}

public string Name {
    get {
        return name;
    }
    set {
        name = value;
```

```
        }
    }

    public IGISResource Resource {
        get {
            return resource;
        }
        set {
            resource = value;
        }
    }

    public bool Initialized {
        get {
            return initialized;
        }
    }

    public void LoadState() {
    }

    public void Initialize() {
        initialized = true;
    }

    public void SaveState() {
    }

    public void Dispose() {
        initialized = false;
    }

    public bool Supports(string operation) {
        return true;
    }

    #endregion
```

对于 Toc 功能来说要求实现的功能相对要少，因此其 GIS 功能接口实现代码也相对简单。在 MapTocFunctionality 类自身代码段中，加入如下代码：

```
    public MapTocFunctionality(string name, MapResource resource) {
        this.name = name;
        this.resource = resource;
        mapResource = resource;
    }

    private MapFunctionality GetMapFunctionality(string name) {
        MapFunctionality mapFunctionality = resource.Functionalities.Find(name)
                      as MapFunctionality;
        return mapFunctionality;
```

```
    }
```

接着要实现的是 Toc 功能接口的方法。Toc 功能接口只规定了一个方法，用于返回一个 TocDataFrame 数组。实现方法如下：

```
    public TocDataFrame[] GetMapContents(string mapFunctionalityName,
    WebImageFormat format, bool useMimeData, bool showAllDataFrames)
    {
        TocDataFrame[] tocDataFrames = new TocDataFrame[1];
        tocDataFrames[0] = new TocDataFrame(resource.Name);
        System.Web.SessionState.HttpSessionState session = null;
        try {
            System.Web.HttpContext httpContext = System.Web.HttpContext.Current;
            if (httpContext != null)
                session = httpContext.Session;
        }
        catch {
        }

        SwatchInfo swatchInfo = new SwatchInfo(10, 10, session);
        foreach (System.Data.DataTable table in mapResource.Graphics.Tables) {
            tocDataFrames[0].Add(GraphicsLayer.GetTocLayer(table as GraphicsLayer,
swatchInfo));
        }
        return tocDataFrames;
    }
```

至此实现了 Toc 功能。

6.2.6　实现地图信息接口

最后要实现的是 IMapInformation 接口。该接口提供只要求实现一些属性，如图 6.7 所示。

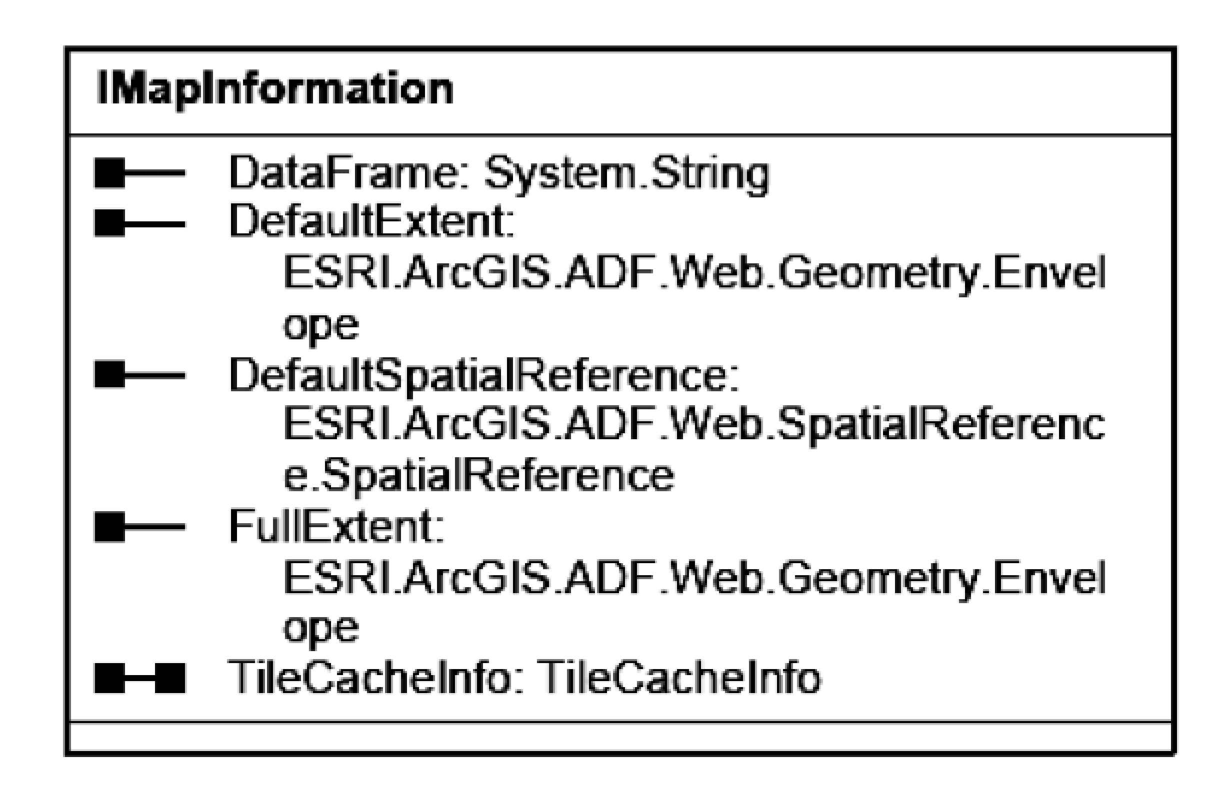


图 6.7　IMapInformation 接口要求实现的属性

在工程中新加一个类，命名为 MapInformation。在其中先加入如下命名空间的引用：

```
    using System.Collections;
```

Web GIS

```
using ESRI.ArcGIS.ADF.Web.DataSources;
using ESRI.ArcGIS.ADF.Web.Display.Graphics;
using ESRI.ArcGIS.ADF.Web.Display.Drawing;
using ESRI.ArcGIS.ADF.Web.Geometry;
using ESRI.ArcGIS.ADF.Web.SpatialReference;
```

将 MapInformation 类的声明修改为如下代码，增加对 IMapInformation 接口实现的声明：

class MapInformation: IMapInformation

该类的实现很简单，代码如下所示：

```
private GraphicsDataSet graphics = null;

public MapInformation(GraphicsDataSet graphics) {
    this.graphics = graphics;
}

#region IMapInformation Members

private string dataFrame = string.Empty;
private SpatialReference defaultSpatialReference = null;
private TileCacheInfo tileCacheInfo = null;

public string DataFrame {
    get {
        return dataFrame;
    }
}

public SpatialReference DefaultSpatialReference {
    get {
        return
            defaultSpatialReference;
    }
}

public Envelope DefaultExtent {
    get {
        if (graphics == null)
            return null;
        else
            return graphics.DefaultExtent;
    }
}

public Envelope FullExtent {
    get {
        if (graphics == null)
            return null;
        else
            return graphics.FullExtent;
```

```
        }
    }

    public TileCacheInfo TileCacheInfo {
        get {
            return tileCacheInfo;
        }
        set {
            tileCacheInfo = value;
        }
    }

    #endregion
```

6.2.7　注册自定义数据源

在地图资源管理器控件的 MapResourcesItem 属性设置对话框的 Map Resource Definition Editor 工具中，Web ADF 已经内置了几个数据源类型，在 Type 选项中有 GraphicsLayer、ArcGIS Server Local、OGC (WMS) Service、ArcIMS、ArcGIS Server Internet、ArcWeb Services 这几种类型可以选择。不过现在要显示自定义的 XML 数据源，就需要为注册新的一个数据源类型来实现。

在 MapResourcesItem 属性设置对话框的 Map Resource Definition Editor 工具中的 DataSource 类型，是通过一个配置文件来实现的，在 ArcGIS.Server 9.2 .Net 中这些配置文件放置在 ArcGIS Server 安装目录的 DotNet 文件内，这里能找到 ArcGIS Server Local、GraphicsLayer、ArcIMS 等所有 DataSource 类型的配置文件。这些都是文本文件，因此读者可以用记事本打这些文件学习一下它们是如何构成的。在 DotNet 文件夹内添加一个名为 ESRI.ArcGIS.ADF.Web.DataSources.XMLData.config 的配置文件，注册我们自定义的数据源。注意该文件名前面必须为 ESRI.ArcGIS.ADF.Web.DataSources。文件所包含的具体内容和说明如下：

```
    <?xml version="1.0" encoding="utf-8" ?>
    <DataSources>
      <!--数据源类型的名称-->
      <DataSource name="XML Data">
        <!--数据源处理类库名以及数据源处理类-->
        <Implementation assembly="XMLDataSource"
                        class="XMLDataSource.GISDataSource">
        </Implementation>
        <IdentityEditorForm assembly="ESRI.ArcGIS.ADF.Web.UI.WebControls"
                            class="ESRI.ArcGIS.ADF.Web.UI.WebControls.Design.Un
usedPropertyEditor">
        </IdentityEditorForm>
        <!--数据源属性编辑器定义-->
        <DefinitionEditorForm assembly="" class=""></DefinitionEditorForm>
        <Resource type="Map">
          <DefinitionEditorForm assembly="" class=""></DefinitionEditorForm>
          <!--地图资源处理库名以及地图资源处理类-->
          <Implementation assembly="XMLDataSource"
```

```
                    class="XMLDataSource.MapResource">
          </Implementation>
        </Resource>
      </DataSource>
    </DataSources>
```

6.2.8 测试自定义 XML 数据源功能

为了验证类库是否成功，我们创建一个简单的 Web 应用程序来测试。

利用 File 菜单的 Add 子菜单的 New Web Site 命令，在当前解决方案中增加一个站点。将其命名为 CutomDataSourceTest。

在 Default.aspx 页面中增加一地图资源管理器控件、一地图控件与一 Toc 控件。打开地图资源管理器控件的 MapResourcesItem 属性设置对话框，在其中先添加一地图资源，然后通过 Definition 属性，打开 Map Resource Definition Editor 对话框。这时在数据源类型的下拉列表框中有了我们在 6.2.7 节中注册的 XML Data 数据源类型（如图 6.8 所示）。选择该类型。然后将 DataSource 设置为我们在 6.2.1 节中文件保存的路径。

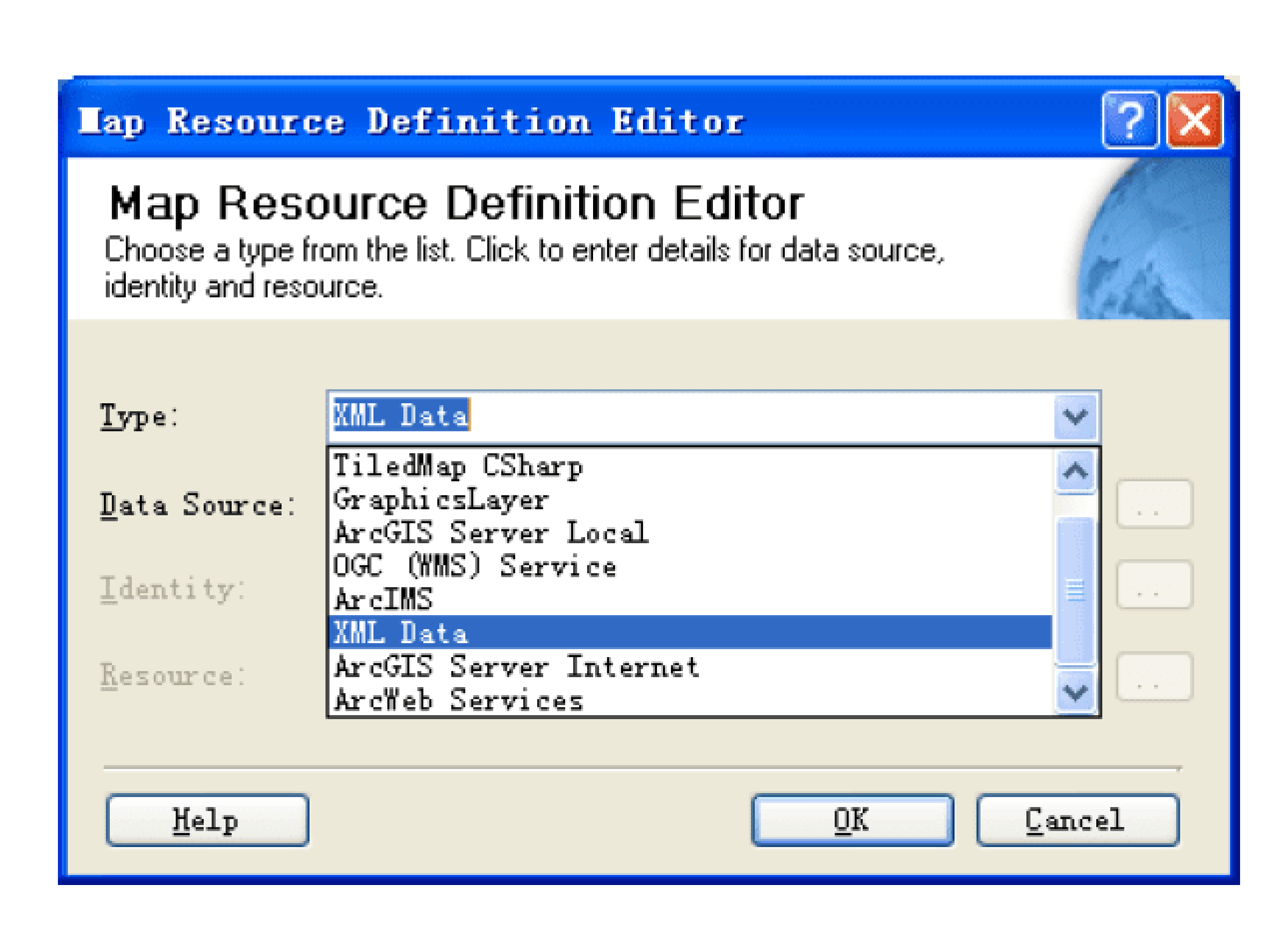


图 6.8 添加自定义数据源

然后再添加 USAMap 地图资源。

由于我们需要利用 XMLDataSource 类库来处理 XML Data 类型的数据集，因此需要在工程中加入该类库对应的动态连接库。一个方法是从 XMLDataSource 工程的 bin 目录下拷贝到本工程的 bin 目录下。另一个更好的方法是直接加入 XMLDataSource 工程引用。可以通过工程的右键菜单的 Add Reference 命令，打开 Add Reference 对话框，切换到 Project 选项卡中，如图 6.9 所示。集成开发环境已经将本解决方案中的其他工程列出了，只需要选择即可。这样一来，当修改了 XMLDataSource 类库后，集成开发环境将最新的类库自动拷贝到本工程的 bin 目录下，而不需要每次都人工操作。

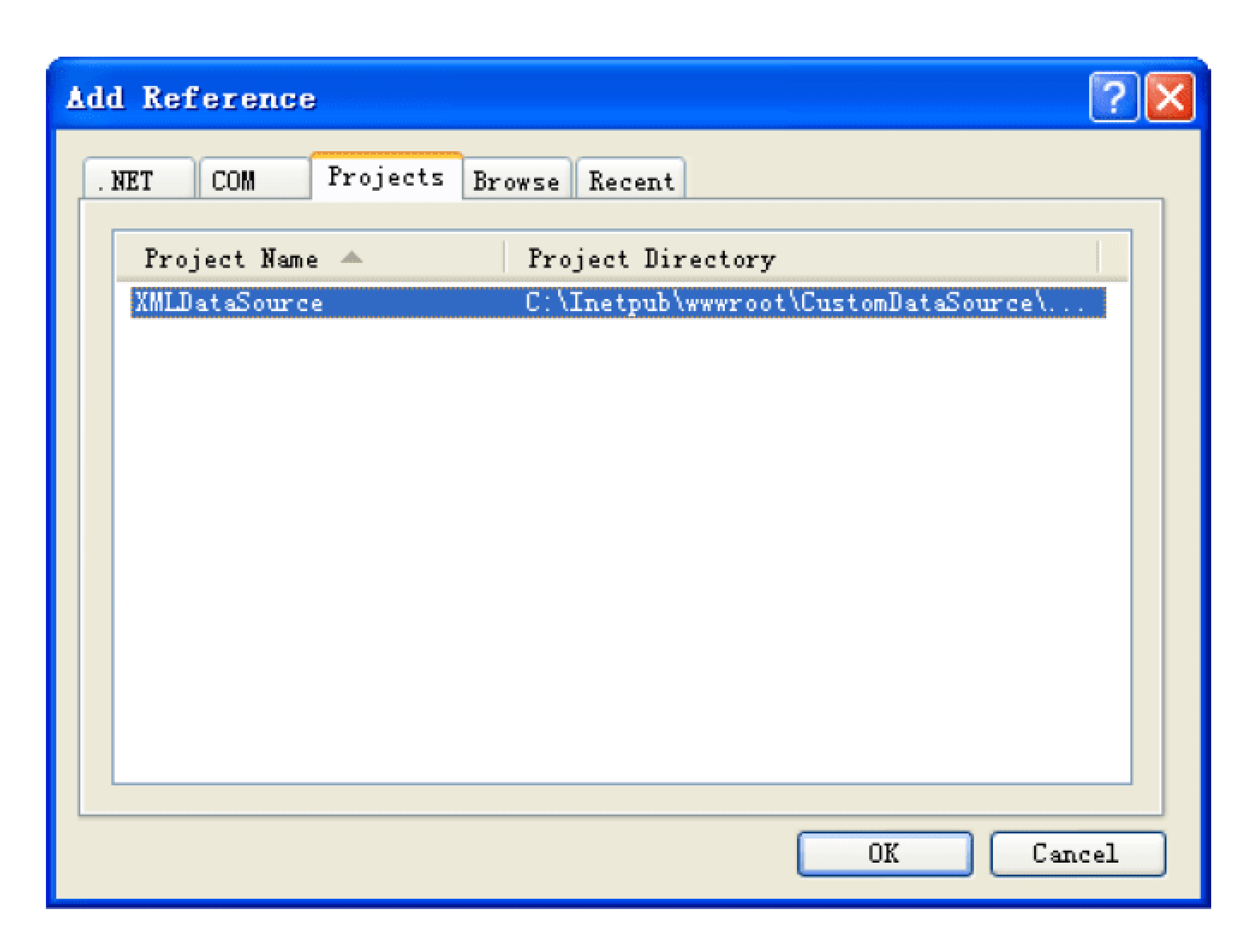


图 6.9　将另一工程作为引用加入本工程

编译并运行程序，程序运行效果如图 6.10 所示，4 个五角星是来自 XML 类型的数据，表明自定义数据源成功。

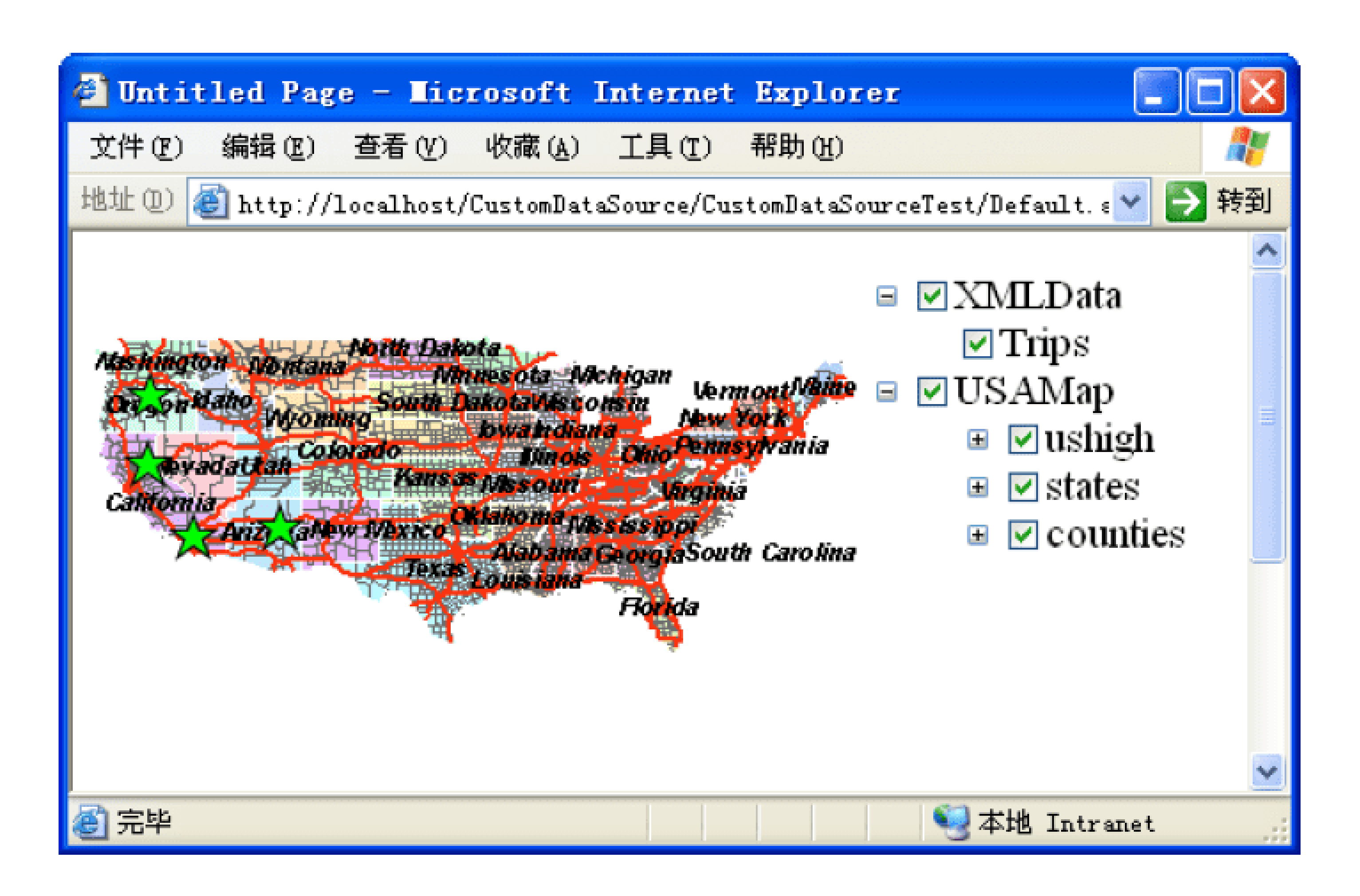


图 6.10　在地图控件中显示自定义的数据源

6.3　遥感影像数据源

在前面的章节中我们介绍的都是矢量要素图层的操作。而在一个实际的地理信息系统中，通常需要用栅格的图像作为背景。在本节中我们将通过把 Virtual Earth 地图图片数据作为数据源，直接使用 MicroSoft 的卫星遥感影像作为系统的背景。

本实例由于使用了地图切片，因此需要实现 ITileFunctionality 接口，在实现该接口中的类中从 Virtual Earth 地图图片网站得到图片即可。

在使用地图切片技术时，地图切片的信息存储在 IMapInformation 接口的 TileCacheInfo 属性中。

地图切片信息对象的 LodInfos 属性存储了一组 LOD（Level Of Details，多细节等级）对象。这些接口与类的关系如图 6.11 所示，弄清楚它们之间的关系是使用地图切片技术的关键。

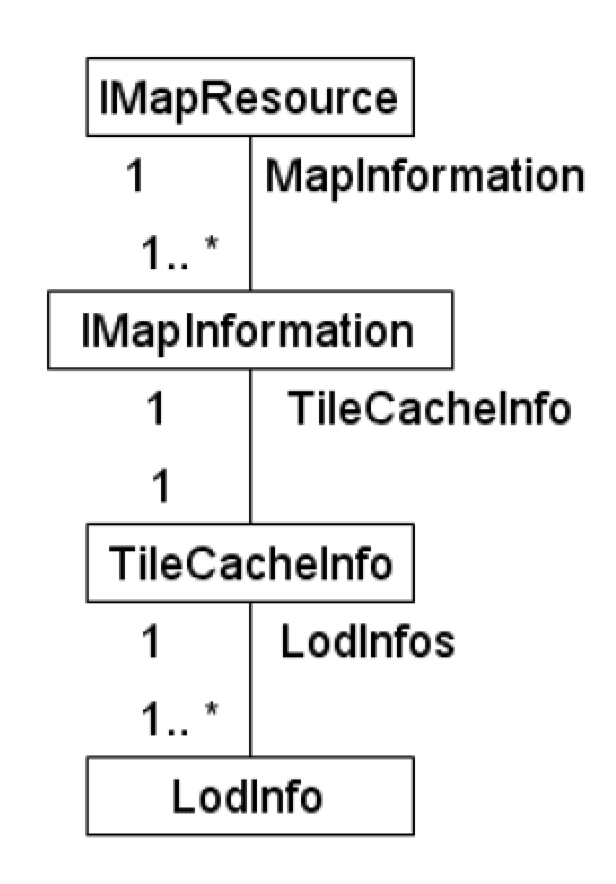


图 6.11 地图切片使用的相关接口与类及其相互之间的关系

利用 File 菜单的 Add 子菜单的 New Project 命令，在 CustomDataSource 解决方案中增加一类库类型的工程，将其命名为 TiledMapDataSource。

通过工程右键菜单的 Add Reference 命令，加入 System.Data、System.Web、ESRI.ArcGIS.ADF、ESRI.ArcGIS.ADF.Web、ESRI.ArcGIS.ADF.Web.DataSource 以及 ESRI.ArcGIS.ADF.Web.UI.WebControls 类库的引用。

6.3.1 实现数据源接口

将新建工程时默认加入的 Class1.cs 文件重新命名为 GISDataSource.cs。

在该类声明代码之前，加入如下命名空间的引用：

```
using System.Web.UI;
using System.Collections;
using ESRI.ArcGIS.ADF.Web.DataSources;
```

修改类声明代码，加入实现 IGISDataSource 接口的声明：

```
public class GISDataSource : IGISDataSource
```

利用集成开发环境的代码生成功能，创建 IGISDataSource 接口要求实现的属性与方法的代码框架，将代码修改为：

```
#region IGISDataSource Members

private string dataSourceDefinition = string.Empty;
private string identity = string.Empty;
bool initialized = false;
private string name = string.Empty;
private Page page = null;
private GISResourceCollection resources = new GISResourceCollection();
```

```
    private Hashtable state;

    /// <summary>
    /// GIS 数据源的名称
    /// </summary>
    public string Name {
        get {
            return name;
        }
        set {
            name = value;
        }
    }

    /// <summary>
    /// 数据源的定义
    /// </summary>
    public string DataSourceDefinition {
        get
        {
            return dataSourceDefinition;
        }
        set {
            if (dataSourceDefinition != value) {
                dataSourceDefinition = value;
            }
        }
    }

    /// <summary>
    /// 连接数据源的身份
    /// </summary>
    public string Identity {
        get {
            return identity;
        }
        set {
            identity = value;
        }
    }

    /// <summary>
    /// 使用该数据源的页面
    /// </summary>
    public Page Page {
        get {
            return page;
        }
        set {
            page = value;
```

```
        }
    }

    /// <summary>
    /// 该数据源能够支持的资源集合
    /// </summary>
    public GISResourceCollection Resources {
        get {
            return resources;
        }
        set {
            resources = value;
        }
    }

    /// <summary>
    /// 数据源是否被初始化
    /// </summary>
    public bool Initialized {
        get {
            return initialized;
        }
    }

    /// <summary>
    /// 状态哈希表
    /// </summary>
    public Hashtable State {
        get {
            return state;
        }
    }

    /// <summary>
    /// 加载状态
    /// </summary>
    /// <param name="state">存储状态的哈希表</param>
    public void LoadState(Hashtable state) {
        this.state = state;
    }

    /// <summary>
    /// 初始化数据源
    /// </summary>
    public void Initialize() {
        initialized = true;
    }

    /// <summary>
    /// 保存状态 Saves state
```

```
/// </summary>
/// <returns>存储状态的哈希表</param>
public Hashtable SaveState() {
    return state;
}

/// <summary>
/// 释放数据源
/// </summary>
public void Dispose() {
    initialized = false;
}

/// <summary>
/// 返回使用该数据源的资源的定义字符串
/// </summary>
public string[] GetAvailableResourceDefinitions(System.Type resourceType) {
    int i = 0;
    string[] definitions = new string[resources.Count];
    foreach (IGISResource rs in resources) {
        definitions[i] = rs.ResourceDefinition;
    }

    return definitions;
}

/// <summary>
/// 根据参数的定义创建资源
/// </summary>
/// <param name="resourceDefinition">要创建的资源的定义</param>
/// <param name="name">要创建的资源的名称</param>
/// <returns>参数指定定义的 GIS 资源</returns>
public IGISResource CreateResource(string resourceDefinition, string name)
{
    throw new Exception("The method or operation is not implemented.");
}

#endregion
```

然后加入 GISDataSource 类的几个构造方法，代码如下：

```
public GISDataSource() {
}

public GISDataSource(string name, string dataSourceDefinition) :
    this(name, string.Empty, dataSourceDefinition) {
}

public GISDataSource(string name, string identity,
string dataSourceDefinition) {
    this.name = name;
```

```
    this.identity = identity;
    this.dataSourceDefinition = dataSourceDefinition;
}
```

通常 GIS 数据源接口的实现类并没有多少功能，主要是定义数据来源的位置。

6.3.2 实现地图资源接口

在工程增加一个类，命名为 MapResource。

在该类声明代码之前加入如下命名空间的引用：

```
using System.Web.UI;
using System.Collections;
using ESRI.ArcGIS.ADF.Web.DataSources;
using ESRI.ArcGIS.ADF.Web.Display.Graphics;
```

将 MapResource 类的声明修改为如下代码，增加对地图资源接口实现的声明：

```
public class MapResource : IMapResource
```

然后利用开发环境的代码自动功能，创建 IMapResource 与 IGISResource 接口要实现的属性与方法的框架。

首先来完成 IMapResource 接口的实现。在其代码段中加入如下代码：

```
private IMapInformation mapInformation = null;
private ESRI.ArcGIS.ADF.Web.DisplaySettings displaySettings = null;

public IMapInformation MapInformation {
    get {
        return mapInformation;
    }
    set {
        mapInformation = value;
    }
}

public ESRI.ArcGIS.ADF.Web.DisplaySettings DisplaySettings {
    get {
        return displaySettings;
    }
    set {
        displaySettings = value;
    }
}
```

IMapResource 接口的实现很简单，先根据要实现的属性，定义两相应的字段，然后加入 get 与 set 方法即可。

在 IGISResource 接口实现代码段中，先实现其要求的属性。代码如下：

```
bool initialized = false;
```

```
    private string name = string.Empty;
    private string resourceDefinition = string.Empty;
    private IGISDataSource dataSource = null;
    private GISFunctionalityCollection functionalities =
        new GISFunctionalityCollection();
    internal Dictionary<string, bool> layerVisibility = null;
    private int validationtimeout = 0;

    public string Name {
        get {
            return name;
        }
        set {
            name = value;
        }
    }

    public int ValidationTimeout {
        get {
            return validationtimeout;
        }
        set {
            validationtimeout = value;
        }
    }

    public string ResourceDefinition {
        get {
            return resourceDefinition;
        }
        set {
            resourceDefinition = value;
        }
    }

    public IGISDataSource DataSource {
        get {
            return dataSource;
        }
        set {
            dataSource = value;
        }
    }

    public GISFunctionalityCollection Functionalities {
        get {
            return functionalities;
        }
        set {
            functionalities = value;
```

```
        }
    }

    public bool Initialized {
        get {
            return initialized;
        }
    }
```

然后在 MapResource 类自身代码段中加入如下构造函数与属性：

```
    public MapResource() {
    }

    public MapResource(string name, GISDataSource dataSource) {
        this.name = name;
        this.dataSource = dataSource;
    }

    private string key {
        get {
            string tmp = this.GetType().ToString() + ":" + name;
            return (tmp);
        }
    }
```

下面要实现的是 IGISResource 接口的方法。其中一类是功能的创建与判断，其中方法 SupportsFunctionality 用于判断当前资源是否支持某类型的功能。对于本实例，支持的是绘图功能、地图切片功能与 Toc 功能，因此实现代码如下：

```
    public bool SupportsFunctionality(System.Type functionalityType) {
        if (functionalityType == typeof(IMapFunctionality))
            return true;
        else if (functionalityType == typeof(ITileFunctionality))
            return true;
        else if (functionalityType == typeof(IMapTocFunctionality))
            return true;
        else return false;
    }
```

CreateFunctionality 方法用于创建指定类型的功能对象。即针对支持的功能，分别用 new 关键字构造 MapFunctionality、TileFunctionality 与 MapTocFunctionality 实例，这些也都是在本节后面内容要实现的类。代码如下：

```
    public IGISFunctionality CreateFunctionality(System.Type functionalityType,
    string functionalityName)
    {
        IGISFunctionality func = null;
        if (functionalityType == typeof(IMapFunctionality)) {
            func = new MapFunctionality(functionalityName, this);
        }
```

```
        else if (functionalityType == typeof(ITileFunctionality)) {
            TileCacheInfo tileCacheInfo =
                    mapInformation.TileCacheInfo as TileCacheInfo;
            func = new TileFunctionality(functionalityName, this, tileCacheInfo);
        }
        else if (functionalityType == typeof(IMapTocFunctionality)) {
            TileCacheInfo tileCacheInfo =
                    mapInformation.TileCacheInfo as TileCacheInfo;
            func = new MapTocFunctionality(functionalityName, this, tileCacheInfo);
        }
        else {
            throw new ArgumentException("functionalityType");
        }
        return func;
    }
```

IGISResource 接口方法的另一大类是有关状态维护的。其中 LoadState 方法用于从数据源对象中获取状态信息，SaveState 方法用于将状态信息保存到数据源对象中，而 ClearState 方法用于清除状态。它们的实现代码如下：

```
    public void LoadState() {
        if (dataSource == null)
            return;
        if (dataSource.State == null)
            return;

        object o = dataSource.State[key];
        if (o != null) {
            MapResource mr = o as MapResource;
            mapInformation = mr.mapInformation;
            layerVisibility = mr.layerVisibility;
        }
    }

    public void SaveState() {
        if (dataSource == null)
            return;
        if (dataSource.State == null)
            return;
        dataSource.State[key] = this;
    }

    public void ClearState() {
        mapInformation = null;
        layerVisibility = null;
    }
```

IGISResource 接口中最重要的方法是初始化方法 Initialize，我们需要在方法中，从数据源指定的位置处读入数据。实现代码如下：

```
    public void Initialize()
```

```
{
    if (mapInformation == null) {
        mapInformation = new MapInformation(dataSource.DataSourceDefinition,
            resourceDefinition);
    }

    if (layerVisibility == null) {
        TileCacheInfo tileCacheInfo =
            (TileCacheInfo)mapInformation.TileCacheInfo;
        layerVisibility = new Dictionary<string, bool>();
        if (tileCacheInfo.Layers != null) {
            foreach (KeyValuePair<string, string> kvp in tileCacheInfo.Layers)
                layerVisibility.Add(kvp.Key, true);
        }
    }
    initialized = true;
}
```

在上述方法中，先利用 MapInformation 类（将在下一小节中实现）的构造函数，从数据源指定的位置处读入地图切片信息，保存在 TileCacheInfo 属性中。然后通过该属性，初始化 layerVisibility 字段。

6.3.3 实现地图信息接口

下面需要实现的是从地图信息接口的类中读取地图切片信息。该切片信息存储在一配置文件中。读者可从本书网上所附源代码文件的 CustomDataSource\CustomDataSourceTest\Data 目录下找到名为 VirtualEarth_original.xml 的文件，即是该配置文件。

在工程增加一个类，命名为 MapInformation。

在该类声明代码之前加入如下命名空间的引用：

```
using ESRI.ArcGIS.ADF.Web.Display.Graphics;
using ESRI.ArcGIS.ADF.Web.Geometry;
using ESRI.ArcGIS.ADF.Web.SpatialReference;
using ESRI.ArcGIS.ADF.Web.DataSources;
using System.Xml;
```

将 MapInformation 类的声明修改为如下代码，增加对地图信息接口实现的声明：

```
public class MapInformation: I MapInformation
```

然后利用开发环境的代码自动功能，创建 MapInformation 接口要实现的属性的框架。代码很简单，纯粹是进行体力劳动，如下所示：

```
private string dataSourceConfig = string.Empty;
private string resourceConfig = string.Empty;
private SpatialReference defaultSpatialReference = null;
private TileCacheInfo tileCacheInfo = null;
private Envelope defaultExtent, fullExtent;
```

```
public string DataFrame {
    get {
        return "(default)";
    }
}

public SpatialReference DefaultSpatialReference {
    get
    {
        return defaultSpatialReference;
    }
}

public Envelope DefaultExtent {
    get {
        return defaultExtent;
    }
}

public Envelope FullExtent {
    get {
        return fullExtent;
    }
}

public ESRI.ArcGIS.ADF.Web.DataSources.TileCacheInfo TileCacheInfo {
    get {
        return tileCacheInfo;
    }
    set {
        tileCacheInfo = new TileCacheInfo(value);
    }
}
```

然后加入 MapInformation 类如下所示的构造函数：

```
    public MapInformation(string dataSourceDefinition, string resourceDefinition)
{
        this.dataSourceConfig = dataSourceDefinition;
        resourceConfig = resourceDefinition;
        parseConfig();
    }
```

在上述构造函数中，调用 parseConfig 方法，从配置文件中读取地图切片信息。该方法的代码如下：

```
private void parseConfig() {
    #region 从文档中得到资源的配置结点
    XmlDocument doc = new XmlDocument();
    // 加载 xml 文档
```

```
        doc.Load(dataSourceConfig);
        XmlNode root = doc.DocumentElement;
        // 读取 TileCacheInfo 节点
        XmlNodeList tileCacheInfoNodes = root.SelectNodes("TileCacheInfo");

        if (tileCacheInfoNodes == null || tileCacheInfoNodes.Count < 1)
            throw new Exception("在资源中找不到配置信息:" + resourceConfig);

        XmlNode tileCacheNode = null;
        Dictionary<int, string> tileUrls = new Dictionary<int, string>();
        Dictionary<string, string> layers = new Dictionary<string, string>();

        for (int t = 0; t < tileCacheInfoNodes.Count; ++t) {
            if        (tileCacheInfoNodes[t].Attributes["Name"].InnerText        ==
resourceConfig) {
                tileCacheNode = tileCacheInfoNodes[t];
                break;
            }
        }
        if (tileCacheNode == null)
            tileCacheNode = tileCacheInfoNodes[0];
        #endregion

        #region 缓存地图切片信息
        // 获取 SpatialReference 节点内容,即坐标系统定义字符串
        string srtext = tileCacheNode.Attributes["SpatialReference"].InnerText;
        int srid;
        // 设置默认的空间参考对象 defaultSpatialReference
        if (Int32.TryParse(srtext, out srid)) {
            defaultSpatialReference = new SpatialReference(srid);
        }
        else {
            defaultSpatialReference = new SpatialReference(srtext);
        }

        int dpi = Convert.ToInt32(tileCacheNode.Attributes["DPI"].InnerText);
        int height =
                Convert.ToInt32(tileCacheNode.Attributes["Height"].InnerText);
        int width = Convert.ToInt32(tileCacheNode.Attributes["Width"].InnerText);
        // 原点坐标
        string[] originCoords = tileCacheNode.Attributes
                            ["TileOrigin"].InnerText.Split(new char[] { ',' });
        double xmin = Convert.ToDouble(originCoords[0]);
        double ymin = Convert.ToDouble(originCoords[1]);
        Point origin = new Point(xmin, ymin);

        // 全图范围
        XmlNode extentNode = tileCacheNode.SelectSingleNode("FullExtent");
        xmin = Convert.ToDouble(extentNode.Attributes["XMin"].InnerText);
        ymin = Convert.ToDouble(extentNode.Attributes["YMin"].InnerText);
```

```
        double xmax = Convert.ToDouble(extentNode.Attributes["XMax"].InnerText);
        double ymax = Convert.ToDouble(extentNode.Attributes["YMax"].InnerText);
        fullExtent = new Envelope(xmin, ymin, xmax, ymax);

        // 默认显示范围
        extentNode = tileCacheNode.SelectSingleNode("DefaultExtent");
        xmin = Convert.ToDouble(extentNode.Attributes["XMin"].InnerText);
        ymin = Convert.ToDouble(extentNode.Attributes["YMin"].InnerText);
        xmax = Convert.ToDouble(extentNode.Attributes["XMax"].InnerText);
        ymax = Convert.ToDouble(extentNode.Attributes["YMax"].InnerText);
        defaultExtent = new Envelope(xmin, ymin, xmax, ymax);
        #endregion

        #region 图层
        // 获取图层 id 和名称
        XmlNode layersNode = tileCacheNode.SelectSingleNode("Layers");
        if (layersNode != null)
        {
            XmlNodeList layerNodes = layersNode.SelectNodes("Layer");
            if (layerNodes != null) {
                for (int l = 0; l < layerNodes.Count; ++l) {
                    string layerId = layerNodes[l].Attributes["LayerID"].InnerText;
                    string layerName = layerNodes[l].Attributes["Name"].InnerText;
                    layers.Add(layerId, layerName);
                }
            }
        }
        #endregion

        #region LOD(Level Of Details，多细节等级，也就是多比例尺)信息
        // 获取各个比例尺的信息
        System.Collections.Generic.List<LodInfo> lodInfos = new List<LodInfo>();
        System.Collections.Generic.Dictionary<int,    int>    levels    =    new
Dictionary<int, int>();
        XmlNode lodInfoNode = tileCacheNode.SelectSingleNode("LodInfos");
        XmlNodeList lodInfoNodes = lodInfoNode.SelectNodes("LodInfo");
        for (int l = 0; l < lodInfoNodes.Count; ++l) {
            // 该细节等级 ID
            int                            levelid                            =
Convert.ToInt32(lodInfoNodes[l].Attributes["LevelID"].InnerText);
            levels.Add(l, levelid);
            // 该细节等级地图切片行方向的数量
            int                              rows                              =
Convert.ToInt32(lodInfoNodes[l].Attributes["Rows"].InnerText);
            // 该细节等级地图切片列方向的数量
            int                            columns                            =
Convert.ToInt32(lodInfoNodes[l].Attributes["Columns"].InnerText);
            // 地图切片 URL 地址，配置文件中没有给出，可以通过程序计算出该地址
            string tileUrl = lodInfoNodes[l].Attributes["TileUrl"].InnerText;
            // 地图切片空间宽度
```

```
            double                                tilewidth                            =
Convert.ToDouble(lodInfoNodes[l].Attributes["TileExtentWidth"].InnerText);
            // 地图切片空间宽度
            double                               tileheight                            =
Convert.ToDouble(lodInfoNodes[l].Attributes["TileExtentHeight"].InnerText);
            // 该细节等级对应的比例尺
            double                                  scale                              =
Convert.ToDouble(lodInfoNodes[l].Attributes["Scale"].InnerText);
            double                               resolution                            =
Convert.ToDouble(lodInfoNodes[l].Attributes["Resolution"].InnerText);
            LodInfo lodInfo = new LodInfo(levelid, resolution, scale, columns, rows,
tilewidth, tileheight);
            lodInfos.Add(lodInfo);
            tileUrls.Add(levelid, tileUrl);
        }
        #endregion

        // 设置 TileCacheInfo
        tileCacheInfo = new TileCacheInfo(width, height,
            dpi, origin, lodInfos.ToArray());
        tileCacheInfo.TileUrls = tileUrls;
        tileCacheInfo.Layers = layers;
        tileCacheInfo.Levels = levels;

        // 获取 Virtual Earth 切片图片路径处理类的类库名称与类名
        XmlNode urlGenNode = tileCacheNode.SelectSingleNode("TileUrlGenerator");
        if (urlGenNode != null)
        {
            tileCacheInfo.TileUrlGeneratorAssembly =
                urlGenNode.Attributes["Assembly"].InnerText;
            tileCacheInfo.TileUrlGeneratorClass =
                urlGenNode.Attributes["Class"].InnerText;
        }
    }
```

代码中已经给出了每段代码的含义，这里不再赘述。

6.3.4 地图切片信息类

为了调用方便，新增加了一个类，继承 Web ADF 中的 TileCacheInfo。该类的属性与方法如图 6.12 所示。

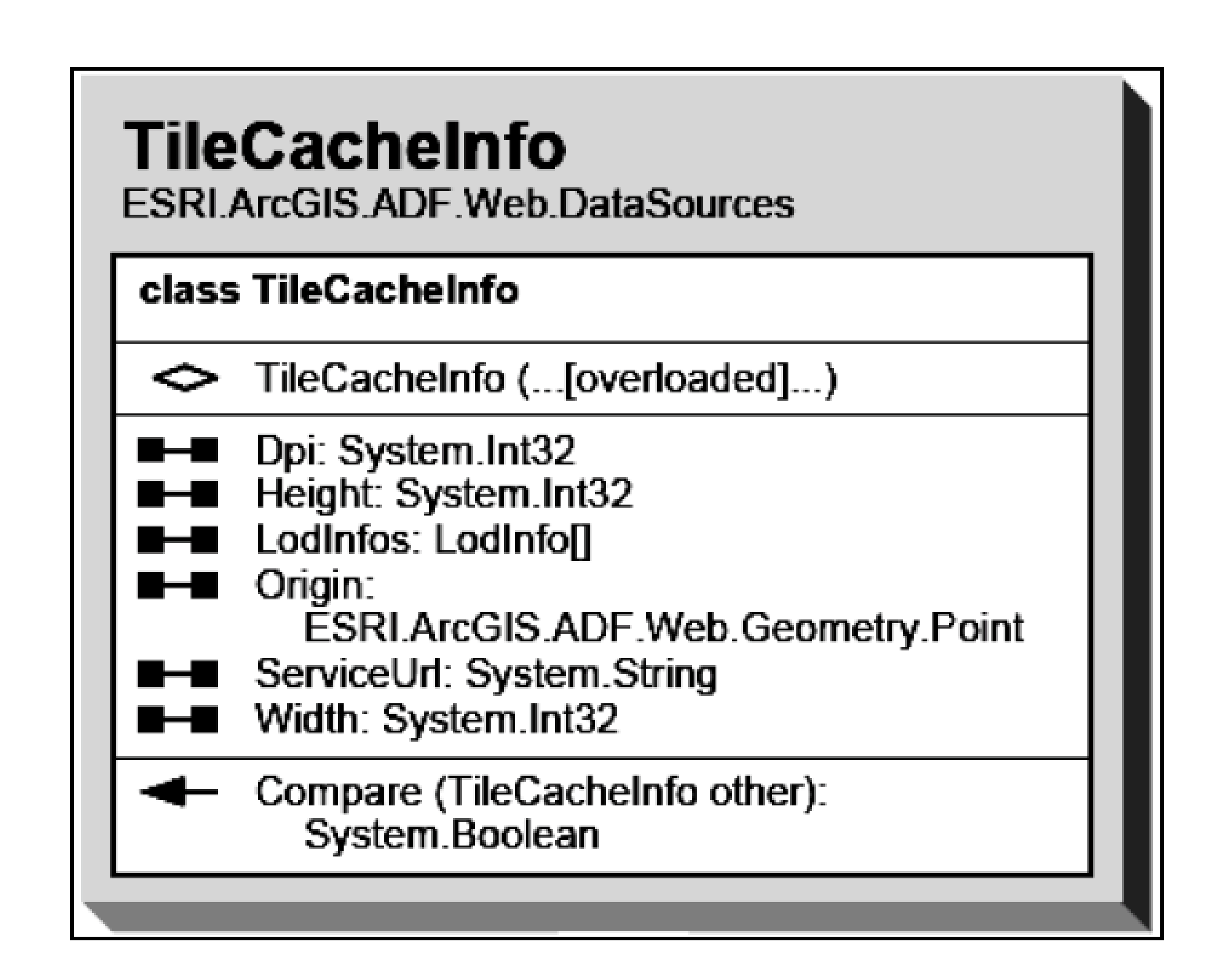


图 6.12　ESRI.ArcGIS.ADF.Web.DataSources.TileCacheInfo 类的属性与方法

在工程增加一个类，命名为 TileCacheInfo。

在该类声明代码之前加入如下命名空间的引用：

```
using ESRI.ArcGIS.ADF.Web.DataSources;
using ESRI.ArcGIS.ADF.Web.Geometry;
```

将 TileCacheInfo 类的声明修改为如下代码：

```
class TileCacheInfo : ESRI.ArcGIS.ADF.Web.DataSources.TileCacheInfo
```

TileCacheInfo 类的代码很简单，主要目的就是增加两个不同的构造函数而已。代码如下：

```
public TileCacheInfo(int width, int height, int dpi,
    Point origin, LodInfo[] lodInfos)
    : base(width, height, dpi, origin, lodInfos) {
}

public TileCacheInfo(ESRI.ArcGIS.ADF.Web.DataSources.TileCacheInfo tci)
    : base(tci.Width, tci.Height, tci.Dpi, tci.Origin, tci.LodInfos) {
}

internal Dictionary<int, int> Levels = new Dictionary<int, int>();
internal Dictionary<int, string> TileUrls = new Dictionary<int, string>();
internal Dictionary<string, string> Layers = new Dictionary<string, string>();
internal string TileUrlGeneratorAssembly;
internal string TileUrlGeneratorClass;
```

6.3.5　实现绘图功能

在工程增加一个类，命名为 MapFunctionality。

在该类声明代码之前加入如下命名空间的引用：

```
using System.Collections;
using ESRI.ArcGIS.ADF.Web.Geometry;
using ESRI.ArcGIS.ADF.Web.SpatialReference;
using ESRI.ArcGIS.ADF.Web.DataSources;
using ESRI.ArcGIS.ADF.Web.Display.Graphics;
using ESRI.ArcGIS.ADF.Web.Display.Drawing;
using ESRI.ArcGIS.ADF.Web;
```

将 MapFunctionality 类的声明修改为如下代码，增加对绘图功能接口实现的声明：

```
public class MapFunctionality: IMapFunctionality
```

然后利用开发环境的代码自动功能，创建 IMapFunctionality 与 IGISFunctionality 接口要实现的属性与方法的框架。

首先来完成 IMapFunctionality 接口的实现。在其代码段中加入如下代码：

```
private ESRI.ArcGIS.ADF.Web.DisplaySettings displaySettings = null;
private System.Web.UI.WebControls.WebControl webControl = null;
private bool maintainsState = false;
private SpatialReference spatialReference = null;
private Dictionary<string, bool> layerVisibility = null;
private Envelope extent;

public bool MaintainsState {
    get {
        return maintainsState;
    }
    set {
        maintainsState = value;
    }
}

public System.Web.UI.WebControls.WebControl WebControl {
    get {
        return webControl;
    }
    set {
        webControl = value;
    }
}

public ESRI.ArcGIS.ADF.Web.DataSources.Units Units {
    get {
        throw new NotImplementedException();
    }
}

public ESRI.ArcGIS.ADF.Web.DisplaySettings DisplaySettings {
    get {
        if (maintainsState) {
            if (displaySettings == null) {
```

```
                displaySettings = MapResource.DisplaySettings.Clone()
                                  as DisplaySettings;
             }
             return displaySettings;
          }
          else
             return MapResource.DisplaySettings;
       }
       set {
          if (maintainsState)
             displaySettings = value;
          else
             MapResource.DisplaySettings = value;
       }
    }

    public object[] LayerIDs {
       get {
          object[] ids = new object[LayerVisibility.Count];
          int i = 0;
          foreach (KeyValuePair<string, bool> kvp in LayerVisibility) {
             ids[i] = kvp.Key.ToString();
             ++i;
          }
          return ids;
       }
    }

    public Envelope Extent {
       get {
          return extent;
       }
       set {
          extent = value;
       }
    }

    public SpatialReference SpatialReference {
       get {
          return spatialReference;
       }
       set {
          spatialReference = value;
       }
    }

    public double GetScale() {
       return double.NaN;
    }
```

```
public double Rotation {
    get {
        return double.NaN;
    }
}

public Dictionary<string, bool> LayerVisibility {
    get {
        if (maintainsState)
            return MapResource.layerVisibility;
        else
            return layerVisibility;
    }
}

public bool GetLayerVisibility(string layerID) {
    return LayerVisibility[layerID];
}

public void SetLayerVisibility(string layerID, bool visible) {
    LayerVisibility[layerID] = visible;
}
```

由于这里使用了地图切片，因此绘图功能并不重要了，不由该类来实现生成地图图片，而是由地图切片功能来实现。该类的主要任务是保存一些地图信息，例如空间参考、显示设置等。因此对于 IMapFunctionality 接口要求实现的其他方法，保留集成开发环境自动生成的代码即可。

在 MapFunctionality 类自身的代码段中，加入如下代码：

```
public MapFunctionality(string name, MapResource resource) {
    this.name = name;
    this.resource = resource;
}

public MapResource MapResource {
    get {
        return resource as MapResource;
    }
}

private string key {
    get {
        string szResource = resource.GetType().ToString() + ":" + resource.Name;
        string szThis = this.GetType().ToString() + ":" + name;
        return (szResource + "," + szThis);
    }
}
```

IGISFunctionality 接口的实现代码如下：

```
private string name = string.Empty;
private IGISResource resource = null;
```

```
    bool initialized = false;

    public string Name {
        get {
            return name;
        }
        set {
            name = value;
        }
    }

    public IGISResource Resource {
        get {
            return resource;
        }
        set {
            resource = value;
        }
    }

    public bool Initialized {
        get {
            return initialized;
        }
    }

    public void LoadState() {
        if (resource.DataSource == null)
            return;
        if (resource.DataSource.State == null)
            return;
        object o = resource.DataSource.State[key];
        if (o != null) {
            MapFunctionality mf = o as MapFunctionality;
            layerVisibility = mf.layerVisibility;
        }
    }

    public void SaveState() {
        if (resource.DataSource == null)
            return;
        if (resource.DataSource.State == null)
            return;
        resource.DataSource.State[key] = this;
    }

    public void Initialize() {
        initialized = true;
        if (layerVisibility == null) {
            layerVisibility = new Dictionary<string, bool>();
```

```
        MapResource mapRes = resource as MapResource;
        foreach (KeyValuePair<string, bool> kvp in mapRes.layerVisibility)
            layerVisibility.Add(kvp.Key, kvp.Value);
    }
}

public void Dispose() {
    initialized = false;
}

public bool Supports(string operation) {
    return false;
}
```

上述代码中，也没有重要的方法。

6.3.6　地图切片功能的实现

Web ADF 中地图切片功能接口的属性与方法如图 6.13 所示，比较简单，就只包含一个生成一张地图图片的 Draw 方法与三个只读属性。

ITileFunctionality

ImageFormat: ESRI.ArcGIS.ADF.Web.WebImageFormat
ServiceUrl: System.String
UrlGeneratorJSFunction: System.String

Draw (System.String mapFunctionalityName, System.Int64 column, System.Int64 row, System.Int32 level): ESRI.ArcGIS.ADF.Web.MapImage

图 6.13　地图切片功能要求实现的属性与方法

在工程增加一个类，命名为 TileFunctionality。

在该类声明代码之前加入如下命名空间的引用：

```
using ESRI.ArcGIS.ADF.Web.DataSources;
using ESRI.ArcGIS.ADF.Web;
using System.Reflection;
```

将 TileFunctionality 类的声明修改为如下代码，增加对地图切片功能接口实现的声明：

```
public class TileFunctionality: ITileFunctionality
```

然后利用开发环境的代码自动功能，创建 ITileFunctionality 与 IGISFunctionality 接口要实现的属性与方法的框架。

首先来完成 IGISFunctionality 接口的实现。在其代码段中加入如下代码：

```
    private string name = string.Empty;
    IGISResource resource = null;
    bool initialized = false;
    System.Web.UI.WebControls.WebControl webControl;

    public System.Web.UI.WebControls.WebControl WebControl {
        get {
            return webControl;
        }
        set {
            webControl = value;
        }
    }

    public string Name {
        get {
            return this.name;
        }
        set {
            this.name = value;
        }
    }

    public IGISResource Resource {
        get {
            return this.resource;
        }
        set {
            this.resource = value;
        }
    }

    public bool Initialized {
        get {
            return initialized;
        }
    }

    public void LoadState() {
    }

    public void SaveState() {
    }

    public void Initialize() {
        initialized = true;
    }

    public void Dispose() {
        initialized = false;
```

```
    }

    public bool Supports(string operation) {
        return true;
    }
```

与绘图功能正好相反，地图切片功能的实现主要是在 ITileFunctionality 中完成，而绘图功能的实现主要在 IGISFunctionality 中完成，因此上述 IGISFunctionality 接口的实现代码非常简单。

在 TileFunctionality 类自身代码段中加入如下代码：

```
    TileCacheInfo tileCacheInfo = null;

    public TileFunctionality(string name, MapResource resource,
    TileCacheInfo tileCacheInfo)
    {
        this.name = name;
        this.resource = resource;
        this.tileCacheInfo = tileCacheInfo;
    }

    private MapFunctionality GetMapFunctionality(string name)
    {
        MapFunctionality mapFunctionality = resource.Functionalities.Find(name)
                                            as MapFunctionality;
        return mapFunctionality;
    }

    private object getInstanceOfType(string assemblyName, string type)
    {
        object o = null;
        try
        {
            Assembly assembly = Assembly.Load(assemblyName);
            o = assembly.CreateInstance(type);
        }
        catch (Exception e)
        {
            string mess = e.Message;
        }
        return o;
    }
```

上述代码中 getInstanceOfType 方法的目的是根据程序集名称与类名，创建类的实例。这是由于是在配置文件中，指定生成 Virtual Earth 地图切片 URL 地址的类，在开发程序时不知道该类的名称，因此需要该方法来在程序运行期间，根据配置参数动态创建类的实例。

下面要实现的是 ITileFunctionality 接口。其代码如下：

```
    private string serviceUrl;

    public string ServiceUrl
```

```
    {
        get
        {
            return serviceUrl;
        }
    }

    public string UrlGeneratorJSFunction
    {
        get
        {
            return null;
        }
    }

    public WebImageFormat ImageFormat
    {
        get
        {
            return WebImageFormat.JPG;
        }
    }

    public ESRI.ArcGIS.ADF.Web.MapImage Draw(string mapFunctionalityName,
    long column, long row, int level)
    {
        int realLevel = tileCacheInfo.Levels[level];
        string baseUrl = tileCacheInfo.TileUrls[realLevel];
        if (string.IsNullOrEmpty(tileCacheInfo.TileUrlGeneratorAssembly)
            || string.IsNullOrEmpty(tileCacheInfo.TileUrlGeneratorClass))
            return null;
        ITileUrlGenerator urlGen = getInstanceOfType
            (tileCacheInfo.TileUrlGeneratorAssembly,
            tileCacheInfo.TileUrlGeneratorClass) as ITileUrlGenerator;
        if (urlGen == null)
            return null;

        MapFunctionality mapFunc = GetMapFunctionality(mapFunctionalityName);
        if (mapFunc == null)
            throw new ArgumentException("mapFunctionalityName");

        serviceUrl = urlGen.GetTileUrl(column, row,
            realLevel, baseUrl, mapFunc.LayerVisibility);

        return new ESRI.ArcGIS.ADF.Web.MapImage(serviceUrl, null);
    }
```

ITileFunctionality 接口中最重要的就是 Draw 方法。在该方法中，我们先利用 getInstanceOfType 方法，创建配置文件中指定的类的事例，然后利用该类的 GetTileUrl 方法，根据地图切片所在的行、列、细节等级，计算地图切片的 URL 地址，最后调用 Web ADF 中 Web 类的 MapImage 静态方法，

从该地址获取一地图图片。

当然在地图切片配置文件中不能随意指定用于获取 URL 地址的类，该类必须实现了 ITileUrlGenerator 接口。

在当前工程中新增加一接口，命名为 ITileUrlGenerator。在其中加入如下代码：

```
public interface ITileUrlGenerator
{
    string GetTileUrl(long column, long row, int level,
        string defaultUrl, Dictionary<string, bool> layerVisibility);
}
```

在该接口定义了一个名为 GetTileUrl 方法，该方法就是利用参数指定的地图切片所在的行、列、细节等级，计算地图切片的 URL 地址。

我们在地图切片配置文件中指定了用 `VirtualEarthTileUrlGenerator` 类来计算 URL。因此还需要实现该类。

在工程中新增加名为 `VirtualEarthTileUrlGenerator` 的类，该类实现 ITileUrlGenerator 接口。

`VirtualEarthTileUrlGenerator` 的实现代码如下：

```
public string GetTileUrl(long column, long row,
    int level, string defaultUrl, Dictionary<string, bool> layerVisibility)
{
    return GetUrl(Convert.ToInt32(column), Convert.ToInt32(row),
        level, layerVisibility);
}

/// <summary>
/// 计算地图切片的 URL 地址
/// </summary>
/// <param name="column">地图切片所在列</param>
/// <param name="row">地图切片所在行</param>
/// <param name="level">地图切片所在细节等级</param>
/// <param name="layerVisibility">可见图层</param>
/// <returns></returns>
private string GetUrl(int column, int row, int level,
    Dictionary<string, bool> layerVisibility)
{
    string mapType = null;
    string mapExtension = null;

    // 如果同时包括 Road 图层和 Aerial 图层
    if (layerVisibility["r"] && layerVisibility["a"])
    {
        mapType = "h";
        mapExtension = ".jpeg";
    }
    //只包括 Road 图层
    else if (layerVisibility["r"])
```

```
        {
            mapType = "r";
            mapExtension = ".png";
        }
        // 只包括 Aerial 图层
        else if (layerVisibility["a"])
        {
            mapType = "a";
            mapExtension = ".jpeg";
        }
        else
            return null;

        string quadKey = TileToQuadKey(column, row, level);

        string url = String.Concat(new object[] { "http://", mapType,
            quadKey[quadKey.Length - 1], ".ortho.tiles.virtualearth.net/tiles/",
            mapType, quadKey, mapExtension, "?g=", 1 });
        return url;
    }

    private static string TileToQuadKey(int tx, int ty, int zl)
    {
        string quad = "";
        for (int i = zl; i > 0; i--)
        {
            int mask = 1 << (i - 1);
            int cell = 0;
            if ((tx & mask) != 0)
            {
                cell++;
            }
            if ((ty & mask) != 0)
            {
                cell += 2;
            }
            quad += cell;
        }
        return quad;
    }
```

对于微软 Virtual Earth 地图切片的 URL 地址计算方法是固定的，具体算法见程序的注释。

6.3.7　实现 Toc 功能

在工程增加一个类，命名为 MapTocFunctionality。

在该类声明代码之前加入如下命名空间的引用：

```
using ESRI.ArcGIS.ADF.Web.DataSources;
```

```
using ESRI.ArcGIS.ADF.Web;
```

将 MapTocFunctionality 类的声明修改为如下代码，增加对绘图功能接口实现的声明：

```
public class MapTocFunctionality: IMapTocFunctionality
```

然后利用开发环境的代码自动功能，创建 IMapTocFunctionality 与 IGISFunctionality 接口要实现的属性与方法的框架。

首先来完成 IGISFunctionality 接口的实现。在其代码段中加入如下代码：

```
private string name = string.Empty;
private IGISResource resource = null;
private bool initialized = false;
System.Web.UI.WebControls.WebControl webControl;

public System.Web.UI.WebControls.WebControl WebControl
{
    get
    {
        return webControl;
    }
    set
    {
        webControl = value;
    }
}

public string Name
{
    get
    {
        return name;
    }
    set
    {
        name = value;
    }
}

public IGISResource Resource
{
    get
    {
        return resource;
    }
    set
    {
        resource = value;
    }
}
```

```
public bool Initialized
{
    get
    {
        return initialized;
    }
}

public void LoadState()
{
}

public void SaveState()
{
}

public void Initialize()
{
    initialized = true;
}

public void Dispose()
{
    initialized = false;
}

public bool Supports(string operation)
{
    return true;
}
```

IGISFunctionality 接口实现的代码很简单，只是简单地设置几个值。

在 MapTocFunctionality 类自身代码段中，加入如下代码：

```
TileCacheInfo _tileCacheInfo = null;
public MapTocFunctionality(string name, MapResource resource,
TileCacheInfo tileCacheInfo)
{
    this.name = name;
    this.resource = resource;
     tileCacheInfo = tileCacheInfo;
}

private MapFunctionality GetMapFunctionality(string name) {
    MapFunctionality mapFunctionality = resource.Functionalities.Find(name)
                              as MapFunctionality;
    return mapFunctionality;
}
```

下面要实现的是 Toc 功能接口。在 IMapTocFunctionality 代码段中加入如下代码：

```
public void SetLayerVisibility(string mapFunctionalityName,
object layerID, bool visible) {
    MapFunctionality mapFunc = GetMapFunctionality(mapFunctionalityName);
    if (mapFunc == null)
        throw new ArgumentException("mapFunctionalityName");
    mapFunc.SetLayerVisibility(layerID.ToString(), visible);
}

public ESRI.ArcGIS.ADF.Web.TocDataFrame[] GetMapContents(
string mapFunctionalityName, WebImageFormat format,
bool useMimeData, bool showAllDataFrames) {
    TocDataFrame[] tocDataFrames = new TocDataFrame[1];
    tocDataFrames[0] = new TocDataFrame(resource.Name);
    if ( tileCacheInfo.Layers != null)
    {
        foreach (KeyValuePair<string, string> kvp in _tileCacheInfo.Layers)
        {
            TocLayer layer = new TocLayer();
            layer.LayerName = kvp.Value;
            layer.ID = kvp.Key;
            layer.Name = kvp.Key;
            layer.Visible = true;
            tocDataFrames[0].Add(layer);
        }
    }
    return tocDataFrames;
}
```

代码也相对简单，在 SetLayerVisibility 方法中，通过调用绘图功能的 SetLayerVisibility 方法设置图层的可见性。在 GetMapContents 方法中，根据切片信息创建 TocLayer。

至此就完成了从微软 Virtual Earth 网站获取地图切片作为数据源的开发。

6.3.8 注册自定义数据源

本实例实现的自定义数据源的配置文件内容如下：

```
<?xml version="1.0" encoding="utf-8" ?>
<DataSources>
   <DataSource name="TiledMap Data">
     <Implementation assembly ="TiledMapDataSource"
      class="TiledMapDataSource.GISDataSource"></Implementation>
     <IdentityEditorForm assembly ="ESRI.ArcGIS.ADF.Web.UI.WebControls"
      class="ESRI.ArcGIS.ADF.Web.UI.WebControls.Design.UnusedPropertyEditor
"></IdentityEditorForm>
    <DefinitionEditorForm assembly ="" class=""></DefinitionEditorForm>
    <Resource type="Map">
       <DefinitionEditorForm assembly ="" class=""></DefinitionEditorForm>
       <Implementation assembly ="TiledMapDataSource"
```

```
            class="TiledMapDataSource.MapResource"></Implementation>
    </Resource>
  </DataSource>
</DataSources>
```

将其保存为 ESRI.ArcGIS.ADF.Web.DataSources.TiledMapData.config，并拷贝到 ArcGIS 安装目录的 DotNet 文件夹中。

该文件读者也可直接从本书网上所附源代码文件的 CustomDataSource\TiledMapDataSource 目录中找到。

6.3.9　测试自定义遥感影像数据源功能

切换到解决方案的 CustomDataSourceTest 工程中，首先通过工程右键菜单的 Add Reference 命令，将 TiledMapDataSource 工程引用进来。

在工程中新增加一个 ASP.NET 页面，命名为 TiledMapTest。

在 TiledMapTest.aspx 页面中增加一地图资源管理器控件、一地图控件与一 Toc 控件。打开地图资源管理器控件的 MapResourcesItem 属性设置对话框，在其中先添加一地图资源，然后通过 Definition 属性，打开 Map Resource Definition Editor 对话框。这时在数据源类型的下拉框中有了我们在 6.3.8 节中注册的数据源类型 TiledMap Data。选择该类型。然后将 DataSource 设置为我们在 6.3.3 节配置文件保存的路径。读者可以直接从本书配套的源代码文件中拷贝该配置文件，文件在 CustomDataSource\CustomDataSourceTest\Data 目录中，文件名为 VirtualEarth_original.xml 。

然后再加入 NorthAmericaMap 地图资源，并加入 ArcGIS 身份信息。

编译并运行程序，便可得到如图 6.14 所示的遥感影像与矢量要素图叠加的效果。

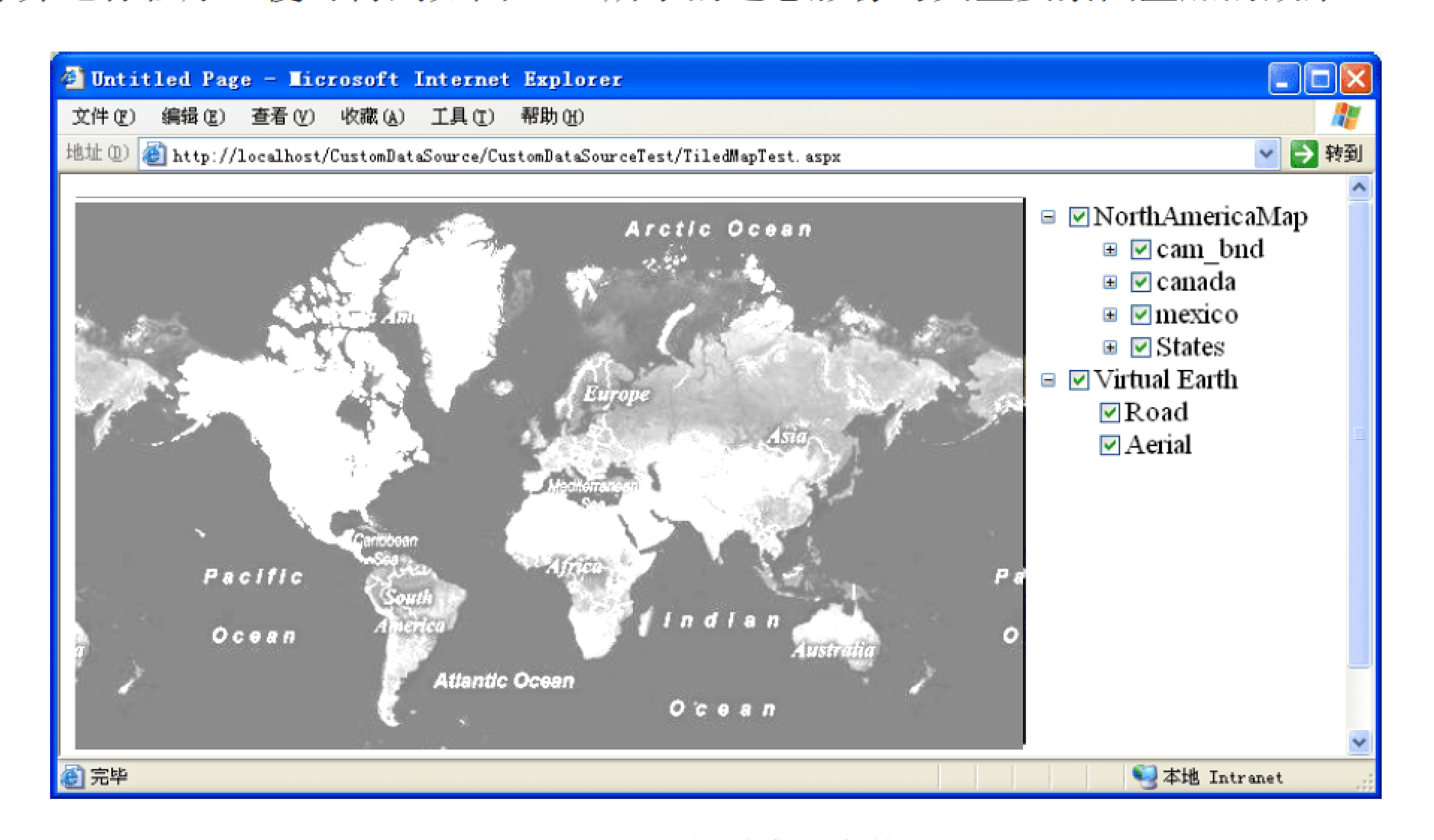


图 6.14　自定义遥感影像数据源

当然互联网上提供类似遥感数据服务的网站很多，包括 Google Earth，以及第 1 章提到的几个网站。有兴趣或需要的读者，也可以通过自定义数据源的方式将它们提供的数据集成到自己的应用程序中。

第 7 章

图形操作

在前面的章节中虽然许多地方都使用了图形及其操作，但是并没有对它们进行过多的说明。我们将在本章中详细介绍图形操作的相关命名空间、类与接口。

通过本章你将了解到：

7.1 图形及相关类概述

正如前面多次提到的，Web ADF 是一多数据源框架，因此可以在同一应用程序中使用多个不同类型的数据。每个数据源还可能独立使用自身特有的 API 来操作数据。由于最终要由 Web ADF 负责将多个数据源的结果显示在 Web 端或客户端，因此 Web ADF 自身必须有一定的逻辑，在一个公共的环境，集成这些结果。为实现该目的，Web ADF 包含了一组类来集成，例如空间参考、几何类型、属性查询以及空间查询等。最重要的是，Web ADF 中有一组类来支持 Web 端图形数据集合与图层的类。

Web ADF 中图形与相关类包含在 ESRI.ArcGIS.ADF.Web.dll 程序集中。该程序集包含根据内容的类别又分成多个命名空间。表 7-1 列出了该程序集中的命名空间及其主要作用。

表 7-1 ESRI.ArcGIS.ADF.Web.dll 程序集中的命名空间及其主要作用

命名空间	主要作用
ESRI.ArcGIS.ADF.Web	Web 端地图图片的创建与管理、空间与属性查询、Toc 图层内容
ESRI.ArcGIS.ADF.Web.Display.Graphics	Web ADF 的图形数据集、图层以及要素
ESRI.ArcGIS.ADF.Web.Display.Renderer	应用于 Web ADF 图形的着色器
ESRI.ArcGIS.ADF.Web.Display.Swatch	用于为 Toc 生成样本
ESRI.ArcGIS.ADF.Web.Display.Symbol	用于 Web ADF 图形的符号
ESRI.ArcGIS.ADF.Web.Geocode	用于地理编码资源与功能的抽象类型
ESRI.ArcGIS.ADF.Web.Geometry	在 Web 端应用的几何类型。常用于为图形图层内容定义要素、地图控件的范围等
ESRI.ArcGIS.ADF.Web.SpatialReference	控件、资源与功能的坐标系统

7.2 图形操作的不同层次

我们在 4.4.3 与 4.5.3 节中使用两种不同的方式了高亮显示通过查询得到的图形，为什么会有两种方式，这两种方式有什么区别呢？

这是由于基于 Web ADF 开发的应用系统包含三层结构，分别是客户端、Web 端以及 GIS 服务器端，因此在 Web ADF 地图中绘制图形可以在三个层次的任何一个层次来实现。每一层的相关开发环境都不一样，因此需要应用不同的方式。由于 Web ADF 的目的是在同一个应用程序中使用多种数据源，因此它提供更多的是在 Web 端创建与管理图形的方法。

非常重要的是，开发人员必须了解通常需要在哪创建图形，以及 Web ADF 是如何集成每个层次的图形的。图 7.1 表明在每个层次上可在哪里创建图形图层。Web ADF 管理包括 Web ADF 图形、ArcGIS Server 与 ArcIMS 等一组资源，其中 Web ADF 图形资源使用 Web ADF 的功能创建图形图层与生成地图图片，这是 4.5.3 节使用的方式。而 ArcGIS Server 与 ArcIMS 资源使用它们各自在 GIS 服务器端的服务功能，来创建图形图层，并与地图中其他图层数据合并生成一张地图图片，这是 4.4.3 节使用的方式。如果将地图控件的 ImageBlendingMode 属性设置为 Browser，这通常是默认值，那么所有的地图图片，依据资源的顺序在浏览器中叠加。此外，客户端浏览器可使用浏览器

的功能来创建图形。

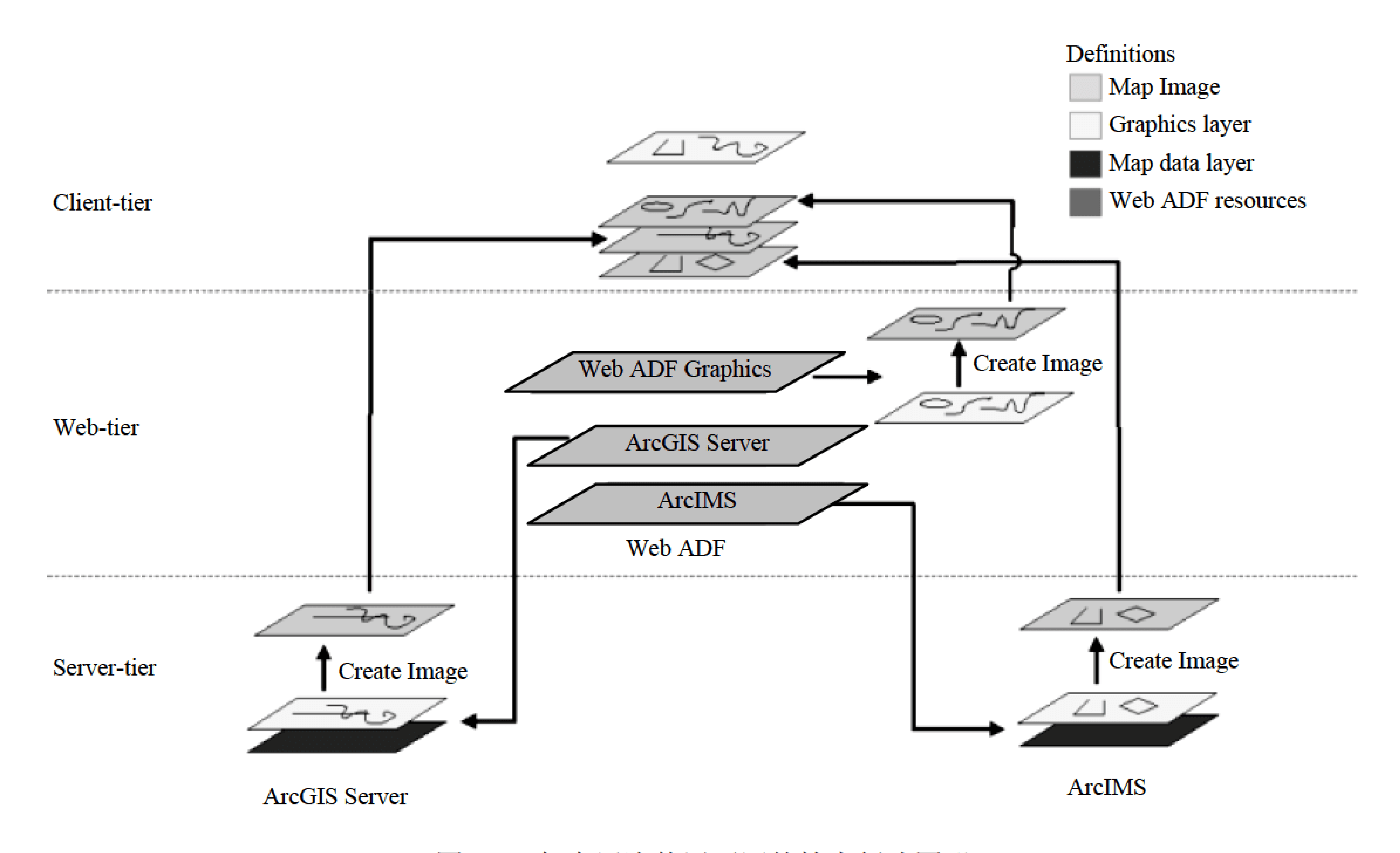


图 7.1 每个层次使用不同的技术创建图形

对于 Web 应用程序来说，客户端大多数情况总是一个浏览器。通常，浏览器使用许多 Web 标准来与显示网页，并与网页交互，这些标准包括 HTML、CSS 以及 JavaScript 等。在该开发环境中，可用工具受限于浏览器的支持。CSS 可用于将图像等元素放置在网页中其他元素之上。浏览器对矢量图形的支持差别很大，但是可以使用 SVG（Scalable Vector Graphics，可缩放矢量图形）与 VML（Vector Markup Language，矢量可标记语言）在网页的其他元素之上绘制矢量图形。

Web ADF 中有许多控件用于客户端的图形。例如 MapTips 控件使用浏览器中的 CSS 在地图上放置一动态提示图像。再例如放大、图形查询工具，需要用户交互时，在地图上绘制矩形、多边形等，在 Internet Explorer 浏览器中利用的是 VML，在其他浏览器中利用的是 CSS。我们可以利用 Web ADF 来帮助在客户端绘制图形，但是功能还是非常有限。

7.3 在 Web 端操作图形

在 4.5.3 节中我们介绍了如何通过在 ElementGraphicsLayer 中加入几何图形对象，从而实现高亮显示查询得到的要素。

ElementGraphicsLayer（几何图形图层）中只包含几何图形对象，即这些对象只有空间位置信息，没有属性信息。但是 Web ADF 中还在 Web 端提供了 FeatureGraphicsLayer（要素图形图层）类，该类中的对象是要素，即既有空间位置信息，又有属性信息。这两个类都是 GraphicsLayer 的子类，而该类又是.NET 中 System.Data.DataTable 类的子类。

我们知道在地图中的数据源是通过资源来访问的，对于 Web 端的图形数据则是通过

ESRI.ArcGIS.ADF.Web.DataSources.Graphics.MapResource 类来访问的。该类的 Graphics 属性的类型是 GraphicsDataSet，它是.NET 中 System.Data.DataSet 类的子类。这就是说 GraphicsDataSet 中包含了 GraphicsLayer 类对象的集合。在 FeatureGraphicsLayer 中又包含了 Geometry 类对象的集合。而 Geometry 是诸如 Envelope、Point、MultiPoint、Polyline 与 Polygon 等这些几何类型对象的父类。这些类之间的关系如图 7.2 所示。

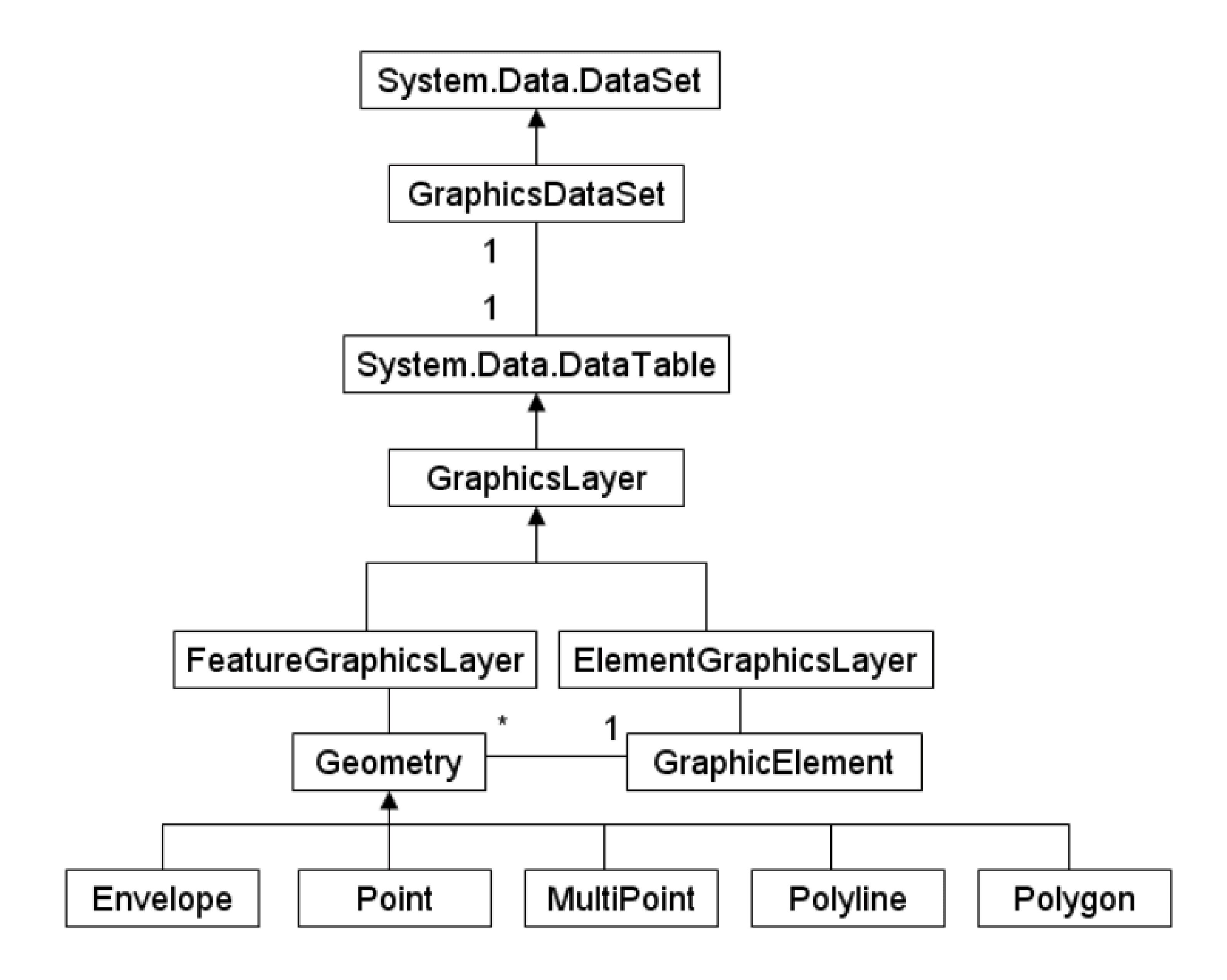


图 7.2　图形对象相关主要类之间的关系

下面我们通过一个实例来介绍如何创建图形对象，如何添加到地图中，以及如何设置符号等。

在 Visual Studio 2005 中，利用 File 菜单的 New Web Site 创建一个新的站点，命名为 CustomRender。

7.3.1　几何对象的创建

几何对象的创建很简单，只需要使用 new 关键字新建 Point、Polyline 与 Polygon 类实例即可。然后通过 FeatureGraphicsLayer 类的 Add 方法加入到图层中即可。

在工程中加入 ASP.NET 文件夹 App_Code，再在其中加入名为 GenerateGraphicsHelper 的类。我们将用在该类中实现创建点、线、多边形要素，并创建要素图形图层。其中点、线、面的空间位置是随机生成的。

在该类的头部加入如下命名空间的引用：

```
using ESRI.ArcGIS.ADF.Web;
using ESRI.ArcGIS.ADF.Web.Geometry;
using ESRI.ArcGIS.ADF.Web.Display.Graphics;
```

然后在类中加入几个辅助的字段：

```
private static Random rand = new Random((int)DateTime.Now.Ticks);
```

```
private static string[] RandomNames = { "Michel Barama",
                              "Richard Gillespie",
                              "Walter Unsworth",
                              "Maurice Burton",
                              "Roy Saunders" };
```

创建点要素图层的方法代码如下：

```
public static FeatureGraphicsLayer CreatePointFeatures(string layerName,
    Envelope env, int count) {
    if (env == null)
        return null;

    FeatureGraphicsLayer layer = new FeatureGraphicsLayer(FeatureType.Point);
    layer.TableName = layerName;
    layer.Columns.Add("imagePath", typeof(string));
    layer.Columns.Add("RandomName", typeof(string));

    for (int i = 0; i < count; i++) {
        double x = env.XMin + rand.NextDouble() * env.Width;
        double y = env.YMin + rand.NextDouble() * env.Height;
        DataRow row = layer.Add(new Point(x, y));
        row["imagePath"] = String.Format("~/images/{0}.gif", rand.Next(3) + 1);
        row["randomName"] = RandomNames[rand.Next(RandomNames.Length)];
    }
    return layer;
}
```

该方法将根据图层名称、范围以及点的个数三个参数，随机创建点要素，并加入到要素图形图层中。在该方法中，先用构造函数新建一个要素图形图层对象，在构造函数的参数中指定要创建的是点要素图层。可以看出，这与几何图形图层不同，在几何图形图层中，可以同时包含点、线、多边形等多类图形。然后利用其 Columns 属性在要素中加入两个属性字段，分别表示图形符号的路径与名称。最后利用一个循环，循环加入点要素。在该循环中，直接用 Point 的构造函数构造点对象，并通过图层对象的 Add 方法返回点要素行对象，然后设置两个属性字段的值。

需要读者在工程中新建 images 文件夹，在其中加入三个小的 gif 图片，分别命名为 1.gif、2.gif 与 3.gif。可以直接从本书配套的源代码文件的 CustomRender 目录中拷贝该文件夹。

创建线要素图形方法的代码如下：

```
public static FeatureGraphicsLayer CreatePolylineFeatures(
string layerName,Envelope env, int count) {
    if (env == null)
        return null;

    FeatureGraphicsLayer layer = new FeatureGraphicsLayer(FeatureType.Line)
    layer.TableName = layerName;
    layer.Columns.Add("Height", typeof(Double));
    layer.Columns.Add("Width", typeof(int));
    layer.Columns.Add("Color", typeof(System.Drawing.Color));
```

```
        double maxsize = env.Width / 10;
        for (int i = 0; i < count; i++) {
            double x = env.XMin + rand.NextDouble() * env.Width;
            double y = env.YMin + rand.NextDouble() * env.Height;
            Path p = new Path();
            p.Points.Add(new Point(x, y));
            for (int j = 0; j < rand.Next(7) + 3; j++) {
                x += rand.NextDouble() * maxsize / 5;
                y += rand.NextDouble() * maxsize / 5 - maxsize / 10;
                p.Points.Add(new Point(x, y));
            }
            Polyline line = new Polyline();
            line.Paths.Add(p);

            DataRow row = layer.Add(line);
            row["Color"] = getRandomColor();
            row["Height"] = rand.NextDouble();
            row["Width"] = Convert.ToInt32(rand.NextDouble() * 5 + 1);
        }
        return layer;
    }
```

代码与创建点要素图层的代码类似，这是多加了几个字段，这几个字段将用于符号的设置。在其中调用的 getRandomColor 方法，得到一个随机的颜色值。该方法的代码如下：

```
    private static System.Drawing.Color getRandomColor() {
        byte[] rgb = new byte[3];
        rand.NextBytes(rgb);
        return System.Drawing.Color.FromArgb(rgb[0], rgb[1], rgb[2]);
    }
```

创建多边形要素图层方法的代码如下：

```
    public static FeatureGraphicsLayer CreatePolygonFeatures(string layerName,
    Envelope env, int count) {
        if (env == null)
            return null;

        FeatureGraphicsLayer layer =
            new FeatureGraphicsLayer(FeatureType.Polygon);
        layer.TableName = layerName;
        layer.Columns.Add("Height", typeof(Double));
        layer.Columns.Add("Width", typeof(int));
        layer.Columns.Add("Color", typeof(System.Drawing.Color));

        double maxsize = env.Width / 10; // 多边形的最大宽度与高度
        for (int i = 0; i < count; i++) {
            double xmin = env.XMin + rand.NextDouble() * env.Width;
            double ymin = env.YMin + rand.NextDouble() * env.Height;
            double rotation = rand.NextDouble() * Math.PI;
            double cosRot = Math.Cos(rotation);
```

```
            double sinRot = Math.Sin(rotation);
            double width = rand.NextDouble() * maxsize;
            double height = rand.NextDouble() * maxsize;
            Ring ring = new Ring();
            ring.Points.Add(new Point(xmin, ymin));
            ring.Points.Add(new Point(xmin + cosRot * width, ymin + sinRot * width));
            ring.Points.Add(new Point(xmin + cosRot * width + sinRot * height,
               ymin + sinRot * width - cosRot * height));
            ring.Points.Add(new Point(xmin + sinRot * height,
                                      ymin - cosRot * height));
            ring.Points.Add(new Point(xmin, ymin));
            Polygon p = new Polygon();
            p.Rings.Add(ring);

            DataRow row = layer.Add(p);
            row["Color"] = getRandomColor(); // 随机颜色
            row["Height"] = rand.NextDouble();
            row["Width"] = Convert.ToInt32(rand.NextDouble() * 5 + 1);
        }
        return layer;
    }
```

创建多边形对象时，需要先创建 Ring 对象，在该对象中加入点对象，然后利用多边形对象的 Rings 属性的 Add 方法加入 Ring 对象。

上面几个方法是随机生成几何要素的空间位置。通常是利用一个查询得到一个图层中所有要素或部分要素，然后构造一个要素图形图层。下面的 AttributeQuery 方法，就用于从 USAMap 地图资源中从美国州级行政区划图层中得到所有要素，对于属性属性，只保留 POP1999（1999 年各州人口数），并按照参数指定分类的个数，进行分类。AttributeQuery 方法的代码如下：

```
    public static FeatureGraphicsLayer AttributeQuery(Map map, int nCount)
    {
        IGISFunctionality gisfunc = map.GetFunctionality("USAMap");
        if (gisfunc == null)
            return null;

        IGISResource gisresource = gisfunc.Resource;
        bool supportquery =
           gisresource.SupportsFunctionality(typeof(IQueryFunctionality));
        if (!supportquery)
            return null;

        IQueryFunctionality qfunc;
        qfunc = gisresource.CreateFunctionality(
                    typeof(IQueryFunctionality), null)
                    as IQueryFunctionality;

        SpatialFilter spatialfilter = new SpatialFilter();
        spatialfilter.ReturnADFGeometries = false;
        spatialfilter.MaxRecords = 100;
```

```
    spatialfilter.WhereClause = "";
    spatialfilter.SubFields.Add("POP1999");

    System.Data.DataTable datatable = qfunc.Query(null, "1", spatialfilter);
    if (datatable == null)
        return null;

    return PolygonFeatureFormTable(map, datatable, nCount);
}

private static FeatureGraphicsLayer PolygonFeatureFormTable(Map map,
DataTable datatable, int nCount) {
    FeatureGraphicsLayer layer = null;
    layer = new FeatureGraphicsLayer(FeatureType.Polygon);
    layer.TableName = datatable.TableName;
    layer.Columns.Add("POP1999", typeof(Double));

    DataRowCollection drs = datatable.Rows;
    int shpind = -1;
    for (int i = 0; i < datatable.Columns.Count; i++) {
        if (datatable.Columns[i].DataType ==
             typeof(ESRI.ArcGIS.ADF.Web.Geometry.Geometry)) {
            // 找到 Geometry 字段的序号
            shpind = i;
            break;
        }
    }

    try {
        double maxValue, minValue;
        MaxMinValue(datatable, "POP1999", out maxValue, out minValue);
        double dist = maxValue - minValue;
        foreach (DataRow dr in datatable.Rows) {
            ESRI.ArcGIS.ADF.Web.Geometry.Geometry geom = dr[shpind]
                as ESRI.ArcGIS.ADF.Web.Geometry.Geometry;

            DataRow row = layer.Add(geom);
            double fieldValue = (double)dr["POP1999"];
            row["POP1999"] = nCount * (fieldValue - minValue) / dist;
        }
    }
    catch (InvalidCastException ice) {
        throw new Exception("No geometry available in datatable");
    }

    return layer;
}

private static void MaxMinValue(System.Data.DataTable datatable,
string field, out double maxValue, out double minValue) {
```

```
        maxValue = 0;
        minValue = 999999999;
        foreach (DataRow dr in datatable.Rows) {
            double fieldValue = (double)dr[field];
            if (maxValue < fieldValue)
                maxValue = fieldValue;
            if (minValue > fieldValue)
                minValue = fieldValue;
        }
    }
```

7.3.2 自定义着色器

在 Web ADF 中，允许为图形图层在默认符号之外，创建自定义的着色器。自定义着色器需要实现 IRender 接口。通过该接口，可以完全利用.NET 中 System.Drawing 命名空间中的类，来控制要素如何在地图中显示。

1. IRender 接口

IRender 接口中包含 5 个方法，但是如果不需要支持 Toc 控件，其实真正需要的只是一个方法，可以使用非常简单的代码实现其他 4 个方法。

这 5 个方法分别是：

- GenerateSwatches——为该着色器在 Toc 控件中的显示创建样本;
- GetAllSymbols——返回该对象使用的所有要素符号对象;
- GetMaxSwatchDimension——得到该着色器创建的图例样本的最大的宽度与高度;
- Clone——返回该着色器的拷贝;
- Render——将几何对象绘制为图形。

在 App_Code 文件夹中新增加一个类，命名为 RendererBase。我们用该类实现最简单的着色器。

先在其头部加入如下代码，引用命名空间：

```
using ESRI.ArcGIS.ADF.Web.Display.Renderer;
using ESRI.ArcGIS.ADF.Web.Geometry;
using ESRI.ArcGIS.ADF.Web.Display.Symbol;
using ESRI.ArcGIS.ADF.Web.Display.Swatch;
using System.Drawing.Drawing2D;
using System.Collections.Generic;
```

将类的声明代码修改为如下：

```
public abstract class RendererBase : IRenderer
```

类的实现代码如下：

```
public virtual SwatchCollection GenerateSwatches(SwatchInfo swatchInfo,
string fileName, string minScale, string maxScale) {
```

```
        return new ESRI.ArcGIS.ADF.Web.Display.Swatch.SwatchCollection();
    }

    public virtual void GetAllSymbols(List<FeatureSymbol> symbols) {
    }

    public virtual void GetMaxSwatchDimensions(ref int width, ref int height) {
    }

    public object Clone() {
        return this.MemberwiseClone() as RendererBase;
    }

    public abstract void Render(DataRow row, System.Drawing.Graphics graphics,
        DataColumn geometryColumn);
```

对于最简单的着色器，可以只实现前 4 个方法。

我们最关心的是 Render 方法，该方法的第一个参数是需要绘制的行，第二个参数是绘制图形对象，第三个参数是几何数据所在字段的名称。

2. 自定义点要素着色器

在 App_Code 文件夹中新加入一类，命名为 SimplePointRenderer。该类将实现用小的 gif 图标来绘制点要素。在该类其头部加入如下命名空间的引用：

```
using ESRI.ArcGIS.ADF.Web;
using ESRI.ArcGIS.ADF.Web.Geometry;
using ESRI.ArcGIS.ADF.Web.Display.Swatch;
using ESRI.ArcGIS.ADF.Web.Display.Symbol;
```

将类的声明修改为如下代码，指定继承 RendererBase 类：

```
public class SimplePointRenderer : RendererBase
```

然后加入一个表示图标路径字段的属性，代码如下：

```
private string imagePathColumn = "ImagePath";
public string ImagePathColumn {
    get {
        return imagePathColumn;
    }
    set {
        imagePathColumn = value;
    }
}
```

该类的 Render 方法代码如下：

```
public override void Render(DataRow row,
    System.Drawing.Graphics graphics, DataColumn geometryColumn) {
    if (row == null || graphics == null || geometryColumn == null)
        return;
```

```
        Geometry geometry = row[geometryColumn] as Geometry;
        if (geometry == null || !(geometry is ESRI.ArcGIS.ADF.Web.Geometry.Point))
            return;

        Point p = geometry as Point;
        if (row.Table.Columns.Contains(imagePathColumn)) {
            string imageIcon = row[imagePathColumn].ToString();
            if
(!System.IO.File.Exists(System.Web.HttpContext.Current.Server.MapPath(imageIco
n))) {
                System.Diagnostics.Debug.Fail("Image path '" + imageIcon + "' not
found");
                return;
            }
            using (System.Drawing.Image img = System.Drawing.Image.FromFile(
                System.Web.HttpContext.Current.Server.MapPath(imageIcon))) {
                graphics.DrawImageUnscaled(img,          Convert.ToInt32(p.X),
Convert.ToInt32(p.Y));
                img.Dispose();
            }
        }
    }
```

在自定义着色器中首先要做的是，得到需要绘制的几何对象。然后利用 System.Drawing 中的类在点的位置绘制一个图标。

Graphics 类提供了 DrawImage 与 DrawImageUnscaled 方法来绘制图像，前者在指定位置并且按指定大小绘制指定的图像，后者在由坐标对指定的位置，使用图像的原始物理大小绘制指定的图像。

支持在 Toc 控件中显示图例的方法的代码如下：

```
    public override SwatchCollection GenerateSwatches(SwatchInfo swatchInfo,
    string fileName, string minScale, string maxScale) {
        SwatchCollection swatches = new SwatchCollection();
        for (int i = 1; i <= 3; i++) {
            CartoImage img = new CartoImage(
                HttpContext.Current.Server.MapPath(
                string.Format("~/images/{0}.gif", i)));
            swatches.Add(new Swatch(img, "Marker #" + i.ToString(), null, null));
        }
        return swatches;
    }
```

在 Default.aspx 页面中增加一地图资源管理器控件、一地图控件与一 Toc 控件。通过地图资源管理器控件的 MapResourcesItem 属性设置对话框，先加入一个 GraphicsLayer 类型的资源，命名为 GraphicsDataSource，然后加入 USAMap 地图资源。

通过工程的右键菜单的 Add ArcGIS Identity 命令，加入连接 USAMap 地图资源的身份信息。

在`_Default`类的头部加入如下命名空间引用：

```
    using ESRI.ArcGIS.ADF.Web.Display.Graphics;
    using ESRI.ArcGIS.ADF.Web.Geometry;
```

```
using ESRI.ArcGIS.ADF.Web.UI.WebControls;
using ESRI.ArcGIS.ADF.Web.DataSources;
```

然后在_Default 类中加入 Page 对象的 PreRender 事件处理方法，代码如下：

```
protected void Page PreRender(object sender, EventArgs e)
{
    if (IsPostBack)
        return;

    MapResourceItem resourceItem =
         MapResourceManager1.ResourceItems.Find("GraphicsDataSource");
    ESRI.ArcGIS.ADF.Web.DataSources.Graphics.MapResource resource =
         resourceItem.Resource as
         ESRI.ArcGIS.ADF.Web.DataSources.Graphics.MapResource;
    if (resource != null)
    {
        FeatureGraphicsLayer layer =
            GenerateGraphicsHelper.CreatePointFeatures(
            "points", Map1.Extent, 50);
        if (layer != null)
        {
            // 应用自定义着色器
            SimplePointRenderer renderer = new SimplePointRenderer();
            renderer.ImagePathColumn = "imagePath"; //  包含图表所在文件的字段名
            layer.Renderer = renderer; //Assign renderer

            //  增加图层
            if (resource.Graphics.Tables.Contains("points"))
                resource.Graphics.Tables.Remove("points");
            resource.Graphics.Tables.Add(layer);
        }
    }
    Map1.RefreshResource("GraphicsDataSource");
}
```

在上述代码中，通过调用 Courier New 类的 CreatePointFeatures 方法，创建一个包含 50 个点要素的图层，然后利用我们自定义的点要素着色器，并把该图层加入到地图资源中。最后调用 RefreshResource 方法来刷新地图。

编译并运行程序，效果如图 7.3 所示。

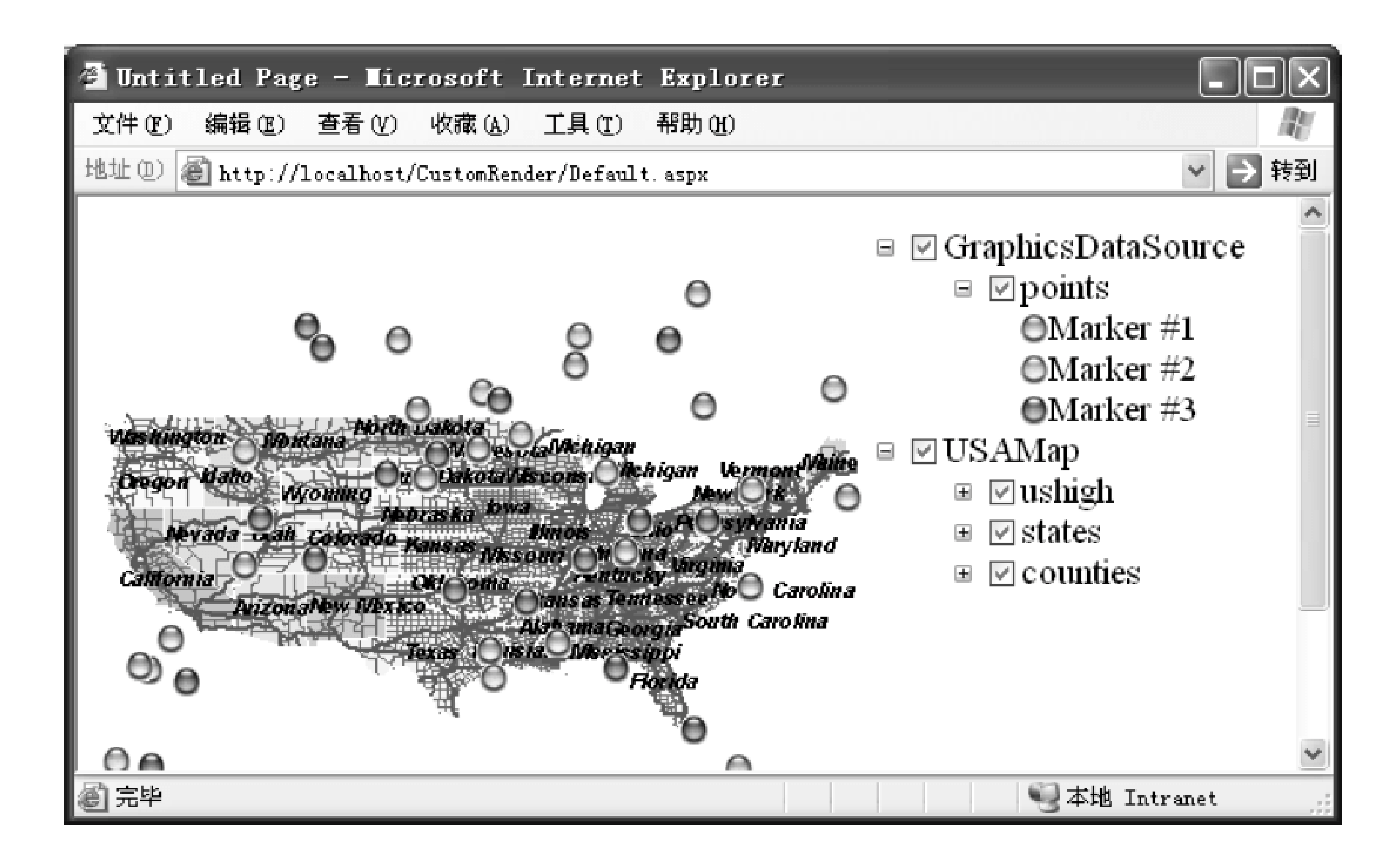


图 7.3 自定义点要素着色器

3. 自定义线要素着色器

在 App_Code 文件夹中新加入一类，命名为 LineColorRenderer。在该类的文件头部加入如下命名空间的引用：

```
using ESRI.ArcGIS.ADF.Web.Geometry;
using ESRI.ArcGIS.ADF.Web.Display.Swatch;
using ESRI.ArcGIS.ADF.Web.Display.Symbol;
```

将类的声明修改为如下代码，指定继承 Courier New 类：

```
public class LineColorRenderer: RendererBase
```

然后加入一表示符号的线的颜色与宽度字段名称的属性，代码如下：

```
private string lineColorColumn = "Color";
public string LineColorColumn
{
    get
    {
        return lineColorColumn;
    }
    set
    {
        lineColorColumn = value;
    }
}

private string lineThicknessColumn = "Width";
public string LineThicknessColumn
{
    get
    {
```

```
            return lineThicknessColumn;
        }
        set
        {
            lineThicknessColumn = value;
        }
    }
```

该类的 Render 方法代码如下：

```
    public override void Render(DataRow row,
        System.Drawing.Graphics graphics, System.Data.DataColumn geometryColumn)
    {
        if (row == null || graphics == null || geometryColumn == null)
            return;
        Geometry geometry = row[geometryColumn] as Geometry;
        if (geometry == null)
            return;

        System.Drawing.Color lineColor = System.Drawing.Color.Black;
        float lineWidth = 1;
        if (row.Table.Columns.Contains(LineThicknessColumn))
        {
            float.TryParse(row[LineThicknessColumn].ToString(), out lineWidth);
        }
        if (row.Table.Columns.Contains(LineColorColumn)
            && row[LineColorColumn] is System.Drawing.Color)
        {
            lineColor = (System.Drawing.Color)row[LineColorColumn];
        }
        if (geometry is Polygon)
            Utility.DrawPolygon(graphics, geometry as Polygon,
                                lineColor, lineWidth);
        else if (geometry is Polyline)
            Utility.DrawPolyline(graphics, geometry as Polyline,
                                lineColor, lineWidth);
    }
```

在该方法中又利用了 Utility 类的两个方法，分别绘制多边形与线。

在 App_Code 文件夹中新加入一类，命名为 Utility。在该类的文件头部加入如下命名空间的引用：

```
    using System.Drawing.Drawing2D;
    using System.Drawing;
```

绘制多边形与线的方法及其相关方法的代码如下：

```
    public static void DrawPolygon(Graphics g,
    ESRI.ArcGIS.ADF.Web.Geometry.Polygon p, Color color, float width)
    {
        DrawPolygon(g, p, color, width, 0, 0);
    }
```

```
    public static void DrawPolygon(Graphics g,
    ESRI.ArcGIS.ADF.Web.Geometry.Polygon p,
    Color color, float width, int offsetX, int offsetY)
    {
        if (p == null)
            return;

        using (GraphicsPath path = Utility.PolygonToPath(p, offsetX, offsetY))
        {
            using (Pen pen = new Pen(color, width))
            {
                g.DrawPath(pen, path);
                pen.Dispose();
            }
            path.Dispose();
        }
    }

    public static GraphicsPath PolygonToPath(
    ESRI.ArcGIS.ADF.Web.Geometry.Polygon polygon)
    {
        return PolygonToPath(polygon, 0, 0);
    }

    public static GraphicsPath PolygonToPath(
    ESRI.ArcGIS.ADF.Web.Geometry.Polygon polygon, int offsetX, int offsetY)
    {
        GraphicsPath path = new GraphicsPath();
        foreach (ESRI.ArcGIS.ADF.Web.Geometry.Ring p in polygon.Rings)
        {
            path.AddPolygon(pointCollectionToPointArray(
                                 p.Points, offsetX, offsetY));
            foreach (ESRI.ArcGIS.ADF.Web.Geometry.Hole ring in p.Holes)
                path.AddPolygon(pointCollectionToPointArray(ring.Points, offsetX,
offsetY));
        }
        return path;
    }

    public static void DrawPolyline(Graphics g,
    ESRI.ArcGIS.ADF.Web.Geometry.Polyline line, Color color, float width)
    {
        if (line == null) return;
        using (GraphicsPath path = Utility.PolylineToPath(line))
        {
            using (Pen pen = new Pen(color, width))
            {
                g.DrawPath(pen, path);
                pen.Dispose();
```

```
            }
            path.Dispose();
        }
    }

    public static GraphicsPath PolylineToPath(
    ESRI.ArcGIS.ADF.Web.Geometry.Polyline polyline)
    {
        GraphicsPath path = new GraphicsPath();
        foreach (ESRI.ArcGIS.ADF.Web.Geometry.Path line in polyline.Paths)
        {
            path.AddLines(pointCollectionToPointArray(line.Points, 0, 0));
        }
        return path;
    }

    private static Point[] pointCollectionToPointArray(
    ESRI.ArcGIS.ADF.Web.Geometry.PointCollection points,
    int offsetX, int offsetY)
    {
        Point[] pointArr = new Point[points.Count];
        for (int i = 0; i < points.Count; i++)
        {
            pointArr[i] = new Point(Convert.ToInt32(points[i].X - offsetX),
                Convert.ToInt32(points[i].Y - offsetY));
        }
        return pointArr;
    }
```

由于在 System.Drawing 中只提供了 DrawPath 方法来绘制 GraphicsPath 类表示路径，因此需要先将 Web ADF 中的 Polyline 与 Polygon 对象转换为路径。

路径可由任意数目的图形（子路径）组成。每一图形都是由一系列相互连接的直线和曲线或几何形状基元构成的。图形的起始点是相互连接的一系列直线和曲线中的第一点。终结点是该序列中的最后一点。几何形状基元的起始点和终结点都是由基元规范定义的。

在工程新加入一 ASP.NET 页面，命名为 CustomLineRender。

在 CustomLineRender.aspx 页面中增加一地图资源管理器控件、一地图控件与一 Toc 控件。通过地图资源管理器控件的 MapResourcesItem 属性设置对话框，先加入一个 GraphicsLayer 类型的资源，命名为 GraphicsDataSource，然后加入 USAMap 地图资源。

在 CustomLineRender 类的头部加入如下命名空间引用：

```
using ESRI.ArcGIS.ADF.Web.Display.Graphics;
using ESRI.ArcGIS.ADF.Web.Geometry;
```

然后在 CustomLineRender 类中加入 Page 对象的 PreRender 事件处理方法，代码如下：

```
protected void Page_PreRender(object sender, EventArgs e)
{
    if (IsPostBack)
        return;
```

```
        MapResourceItem resourceItem =
            MapResourceManager1.ResourceItems.Find("GraphicsDataSource");
        MapResource resource = resourceItem.Resource as MapResource;
        if (resource != null)
        {
            // 先加入一多边形图层
            FeatureGraphicsLayer layer =
                GenerateGraphicsHelper.CreatePolygonFeatures(
                "polygons", Map1.Extent, 10);
            addLayer(resource, layer);
            //  然后加入线图层
            layer = GenerateGraphicsHelper.CreatePolylineFeatures(
                                          "polylines", Map1.Extent, 10);
            addLayer(resource, layer);
        }
        Map1.RefreshResource("GraphicsDataSource");
    }

    private static void addLayer(
    ESRI.ArcGIS.ADF.Web.DataSources.Graphics.MapResource resource,
    FeatureGraphicsLayer layer)
    {
        if (layer != null)
        {
            // 应用着色器
            LineColorRenderer renderer = new LineColorRenderer();
            renderer.LineColorColumn = "Color"; //  包含线的颜色信息的字段名
            renderer.LineThicknessColumn = "Width"; //  线的宽度的字段名
            layer.Renderer = renderer;

            // 增加图层
            if (resource.Graphics.Tables.Contains(layer.TableName))
                resource.Graphics.Tables.Remove(layer.TableName);
            resource.Graphics.Tables.Add(layer);
        }
    }
```

编译并运行程序，程序运行效果如图 7.4 所示。

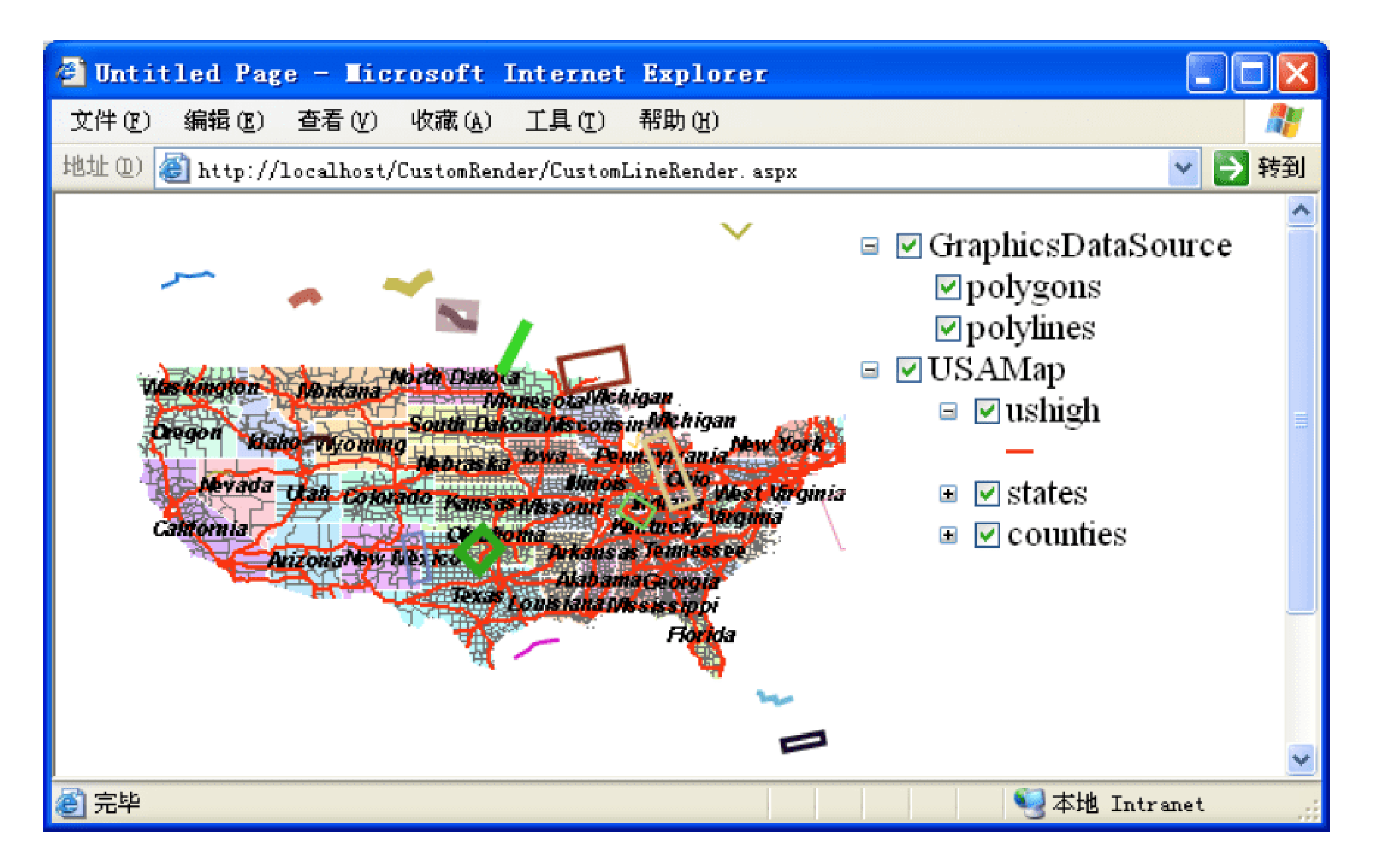


图 7.4　自定义线要素着色器

4. 注记着色器

注记着色器就是要在地图上绘制文本信息。

在 App_Code 文件夹中新加入一类，命名为 LabelPointRenderer。在该类其头部加入如下命名空间的引用：

```
using ESRI.ArcGIS.ADF.Web.Geometry;
using ESRI.ArcGIS.ADF.Web.Display.Swatch;
using ESRI.ArcGIS.ADF.Web.Display.Symbol;
```

将类的声明修改为如下代码，指定继承 RendererBase 类：

```
public class LabelPointRenderer: RendererBase
```

然后加入表示背景颜色、字体等的属性，代码如下：

```
public LabelPointRenderer() : base()
{
    Font = new System.Drawing.Font("Arial", 8f);
}

private string labelColumn = "Name";
public string LabelColumn
{
    get
    {
        return labelColumn;
    }
    set
    {
        labelColumn = value;
    }
}
```

```
private System.Drawing.Color backgroundColor = System.Drawing.Color.White;
public System.Drawing.Color BackgroundColor
{
    get
    {
        return backgroundColor;
    }
    set
    {
        backgroundColor = value;
    }
}

private System.Drawing.Color outlineColor = System.Drawing.Color.Black;
public System.Drawing.Color OutlineColor
{
    get
    {
        return outlineColor;
    }
    set
    {
        outlineColor = value;
    }
}

private System.Drawing.Color fontColor = System.Drawing.Color.Black;
public System.Drawing.Color FontColor
{
    get
    {
        return fontColor;
    }
    set
    {
        fontColor = value;
    }
}

private System.Drawing.Font font;
public System.Drawing.Font Font
{
    get
    {
        return font;
    }
    set
    {
        font = value;
```

```
        }
    }
```

Render 方法的代码如下：

```
    public override void Render(System.Data.DataRow row,
    System.Drawing.Graphics graphics, System.Data.DataColumn geometryColumn)
    {
        if (row == null || graphics == null || geometryColumn == null)
            return;
        Geometry geometry = row[geometryColumn] as Geometry;
        if (geometry == null || !(geometry is ESRI.ArcGIS.ADF.Web.Geometry.Point))
            return;

        ESRI.ArcGIS.ADF.Web.Geometry.Point p =
            geometry as ESRI.ArcGIS.ADF.Web.Geometry.Point;
        if (row.Table.Columns.Contains(LabelColumn))
        {
            string text = row[LabelColumn].ToString();
            // 得到文本的大小
            System.Drawing.SizeF size = graphics.MeasureString(text, Font);
            int margin = 2; // 文本周围的边距
            int arrowSize = 5; // 指向点坐标的箭头的大小
            int width = (int)size.Width + margin * 2;
            int height = (int)size.Height + margin * 2;

            System.Drawing.Drawing2D.GraphicsPath path =
                new System.Drawing.Drawing2D.GraphicsPath()
            int x = (int)p.X;
            int y = (int)p.Y;
            System.Drawing.Point[] pnts = new System.Drawing.Point[8];
            pnts[0] = new System.Drawing.Point(x, y); // 起点
            pnts[1] = new System.Drawing.Point(x - arrowSize, y - arrowSize);
            pnts[2] = new System.Drawing.Point(x - width / 2, pnts[1].Y);
            pnts[3] = new System.Drawing.Point(pnts[2].X, pnts[2].Y - height);
            pnts[4] = new System.Drawing.Point(pnts[3].X + width, pnts[3].Y);
            pnts[5] = new System.Drawing.Point(pnts[4].X, pnts[4].Y + height);
            pnts[6] = new System.Drawing.Point(x + arrowSize, y - arrowSize);
            pnts[7] = new System.Drawing.Point(x, y); // 终点
            path.AddPolygon(pnts);
            // 绘制背景
            graphics.FillPath(new    System.Drawing.SolidBrush(BackgroundColor),
path);
            // 绘制边线
            graphics.DrawPath(new System.Drawing.Pen(OutlineColor), path);
            // 绘制属性
            graphics.DrawString(text, Font,
                new System.Drawing.SolidBrush(FontColor),
                new System.Drawing.PointF(x - width / 2 + margin, y - height -
arrowSize + margin));
        }
```

```
}
```

Graphics 类的 DrawString 方法用于绘制文本。这个函数有多个重载版本，但是它们都需要以下 4 个元素：绘制的字符串（String）、使用的字体对象（Font）、为填充字体所要使用的刷子对象（Brush）以及文本绘制的位置（PointF），这个位置可以通过一个 X 坐标与一个 Y 坐标、一个点或者是一个有边界的矩形表示。

在工程新加入一 ASP.NET 页面，命名为 LaberRenderer。

在 LaberRenderer.aspx 页面中增加一地图资源管理器控件、一地图控件与一 Toc 控件。通过地图资源管理器控件的 MapResourcesItem 属性设置对话框，先加入一个 GraphicsLayer 类型的资源，命名为 GraphicsDataSource，然后加入 USAMap 地图资源。

在 LabelRenderer 类的头部加入如下命名空间引用：

```
using ESRI.ArcGIS.ADF.Web.Display.Graphics;
using ESRI.ArcGIS.ADF.Web.Geometry;
using ESRI.ArcGIS.ADF.Web.UI.WebControls;
using ESRI.ArcGIS.ADF.Web.DataSources.Graphics;
```

然后在 LabelRenderer 类中加入 Page 对象的 PreRender 事件处理方法，代码如下：

```
protected void Page PreRender(object sender, EventArgs e)
{
    if (IsPostBack)
        return;

    MapResourceItem resourceItem =
        MapResourceManager1.ResourceItems.Find("GraphicsDataSource");
    MapResource resource = resourceItem.Resource as MapResource;
    if (resource != null)
    {
        FeatureGraphicsLayer layer =
                 GenerateGraphicsHelper.CreatePointFeatures(
                 "points", Map1.Extent, 20);
        if (layer != null)
        {
            // 应用着色器
            LabelPointRenderer renderer = new LabelPointRenderer();
            renderer.LabelColumn = "RandomName";
            renderer.BackgroundColor =
                System.Drawing.Color.FromArgb(255, 255, 255, 200);
            layer.Renderer = renderer;

            // 增加图层
            if (resource.Graphics.Tables.Contains("points"))
                resource.Graphics.Tables.Remove("points");
            resource.Graphics.Tables.Add(layer);
        }
    }
    Map1.RefreshResource("GraphicsDataSource");
}
```

编译并运行程序，运行效果如图 7.5 所示。

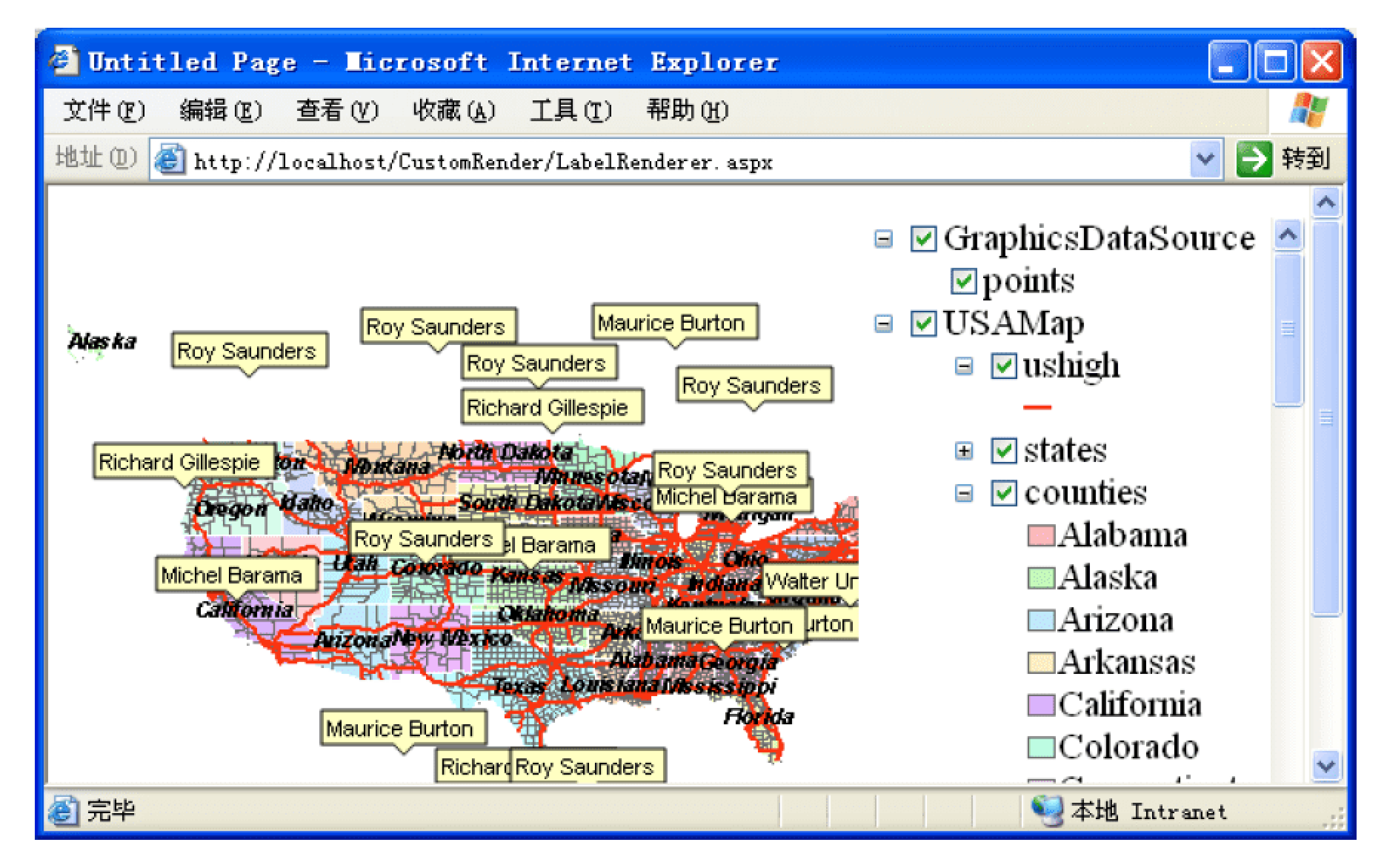


图 7.5　自定义注记着色器

5. 自定义范围专题图着色器

在一个专题图中，范围专题图着色器将数值集合成一个个离散域，并且为每个域指定一个符号。下面是一个描述某地人口数量的例子，假设要做一张地图，并且想用五种不同颜色描述人口数的五个值域：0～9999 人用黄色标注；10000～49999 人的用浅橙色；50000～99999 人的用橙色；100000 人以上的用红色。

在 App_Code 文件夹中新加入一类，命名为 GraduatedColorRenderer。在该类其头部加入如下命名空间的引用：

```
using ESRI.ArcGIS.ADF.Web.Display.Renderer;
using ESRI.ArcGIS.ADF.Web.Geometry;
using ESRI.ArcGIS.ADF.Web.Display.Symbol;
using System.Drawing.Drawing2D;
using System.Collections.Generic;
using ESRI.ArcGIS.ADF.Web.Display.Swatch;
```

将类的声明修改为如下代码，指定继承 RendererBase 类：

```
public class GraduatedColorRenderer: RendererBase
```

然后加入一表示开始颜色、终止颜色等属性，代码如下：

```
private System.Drawing.Color startColor = System.Drawing.Color.Red;
public System.Drawing.Color StartColor
{
    get
    {
        return startColor;
    }
    set
```

```
        {
            startColor = value;
        }
    }

    private System.Drawing.Color endColor = System.Drawing.Color.Yellow;
    public System.Drawing.Color EndColor
    {
        get
        {
            return endColor;
        }
        set
        {
            endColor = value;
        }
    }

    private double minValue = 0;
    public double MinValue
    {
        get
        {
            return minValue;
        }
        set
        {
            minValue = value;
        }
    }

    private double maxValue = 100;
    public double MaxValue
    {
        get
        {
            return maxValue;
        }
        set
        {
            maxValue = value;
        }
    }

    private string colorColumnName;
    public string ColorColumnName
    {
        get
        {
            return colorColumnName;
```

```
        }
        set
        {
            colorColumnName = value;
        }
    }
```

Render 方法及其服务方法的代码如下：

```
    public override void Render(DataRow row, System.Drawing.Graphics graphics,
        DataColumn geometryColumn)
    {
        if (row == null || graphics == null || geometryColumn == null)
            return;

        Geometry geometry = row[geometryColumn] as Geometry;
        if (geometry == null || geometry is ESRI.ArcGIS.ADF.Web.Geometry.Point)
            return;

        if (!string.IsNullOrEmpty(ColorColumnName)
            && row.Table.Columns.Contains(ColorColumnName))
        {
            double value = 0;
            if (double.TryParse(row[ColorColumnName].ToString(), out value))
            {
                System.Drawing.Color color = interpolateColor(value);
                if (geometry is Polygon)
                {
                    Utility.FillPolygon(graphics, geometry as Polygon, color, 0, 0,
0);
                }
                else if (geometry is Polyline)
                {
                    Utility.DrawPolyline(graphics, geometry as Polyline, color, 2);
                }
            }
        }
    }

    private System.Drawing.Color interpolateColor(double value)
    {
        if (value <= minValue)
            return startColor;
        if (value >= maxValue)
            return endColor;
        double frac = (value - minValue) / (maxValue - minValue);

        int r = (int)Math.Round(startColor.R * (1 - frac) + endColor.R * (frac));
        int g = (int)Math.Round(startColor.G * (1 - frac) + endColor.G * (frac));
        int b = (int)Math.Round(startColor.B * (1 - frac) + endColor.B * (frac));
        int a = (int)Math.Round(startColor.A * (1 - frac) + endColor.A * (frac));
```

```
    return System.Drawing.Color.FromArgb(a, r, g, b);
}
```

在上面的代码中通过调用 Utility 类的 FillPolygon 与 DrawPolyline 来绘制每个要素的。DrawPolyline 方法已经在前面就已经加入了，FillPolygon 方法及其辅助方法的代码如下：

```
public static void FillPolygon(Graphics g,
ESRI.ArcGIS.ADF.Web.Geometry.Polygon p,
Color color, int transparency, int offsetX, int offsetY)
{
    if (p == null)
        return;

    using (GraphicsPath path = Utility.PolygonToPath(p, offsetX, offsetY))
    {
        Color fillColor =
            Color.FromArgb(TransparencyToAlpha(transparency), color);
        using (SolidBrush brush = new SolidBrush(fillColor))
        {
            g.FillPath(brush, path);
            brush.Dispose();
        }
        path.Dispose();
    }
}

public static byte TransparencyToAlpha(double percentage)
{
    double value = 0;
    value = 100 - percentage;
    value = value / 100;
    value = value * 255;
    if (value < 0)
        value = 0;
    else if (value > 255)
        value = 255;
    return (byte)value;
}
```

由于这里绘制多边形需要的是填充效果，因此调用的是 Graphics 类的 FillPath 方法，填充 GraphicsPath 的内部。

在工程新加入一 ASP.NET 页面，命名为 GraduatedRenderer。

在 GraduatedRenderer.aspx 页面中增加一地图资源管理器控件、一地图控件与一 Toc 控件。通过地图资源管理器控件的 MapResourcesItem 属性设置对话框，先加入一个 GraphicsLayer 类型的资源，命名为 GraphicsDataSource，然后加入 USAMap 地图资源。

在 GraduatedRenderer 类的头部加入如下命名空间引用：

```
using ESRI.ArcGIS.ADF.Web.Display.Graphics;
using ESRI.ArcGIS.ADF.Web.Geometry;
using ESRI.ArcGIS.ADF.Web.UI.WebControls;
```

```
using ESRI.ArcGIS.ADF.Web.DataSources.Graphics;
```

然后在 `GraduatedRenderer` 类中加入 Page 对象的 PreRender 事件处理方法，代码如下：

```
protected void Page PreRender(object sender, EventArgs e)
{
    if (IsPostBack)
        return;

    MapResourceItem resourceItem =
        MapResourceManager1.ResourceItems.Find("GraphicsDataSource");
    MapResource resource = resourceItem.Resource as MapResource;
    if (resource != null)
    {
        FeatureGraphicsLayer layer =
          GenerateGraphicsHelper.AttributeQuery(Map1, 5);
        addLayer(resource, layer);
    }
    Map1.RefreshResource("GraphicsDataSource");
}

private static void addLayer(MapResource resource, FeatureGraphicsLayer layer)
{
    if (layer != null)
    {
        // 应用着色器
        GraduatedColorRenderer renderer = new GraduatedColorRenderer();
        renderer.ColorColumnName = "POP1999"; // 用于插值的字段名称
        renderer.StartColor = System.Drawing.Color.Blue;
        renderer.EndColor = System.Drawing.Color.Red;
        renderer.MinValue = 1;
        renderer.MaxValue = 6;
        layer.Renderer = renderer;

        // 增加图层
        if (resource.Graphics.Tables.Contains(layer.TableName))
            resource.Graphics.Tables.Remove(layer.TableName);
        resource.Graphics.Tables.Add(layer);
    }
}
```

编译并运行程序，运行效果如图 7.6 所示。

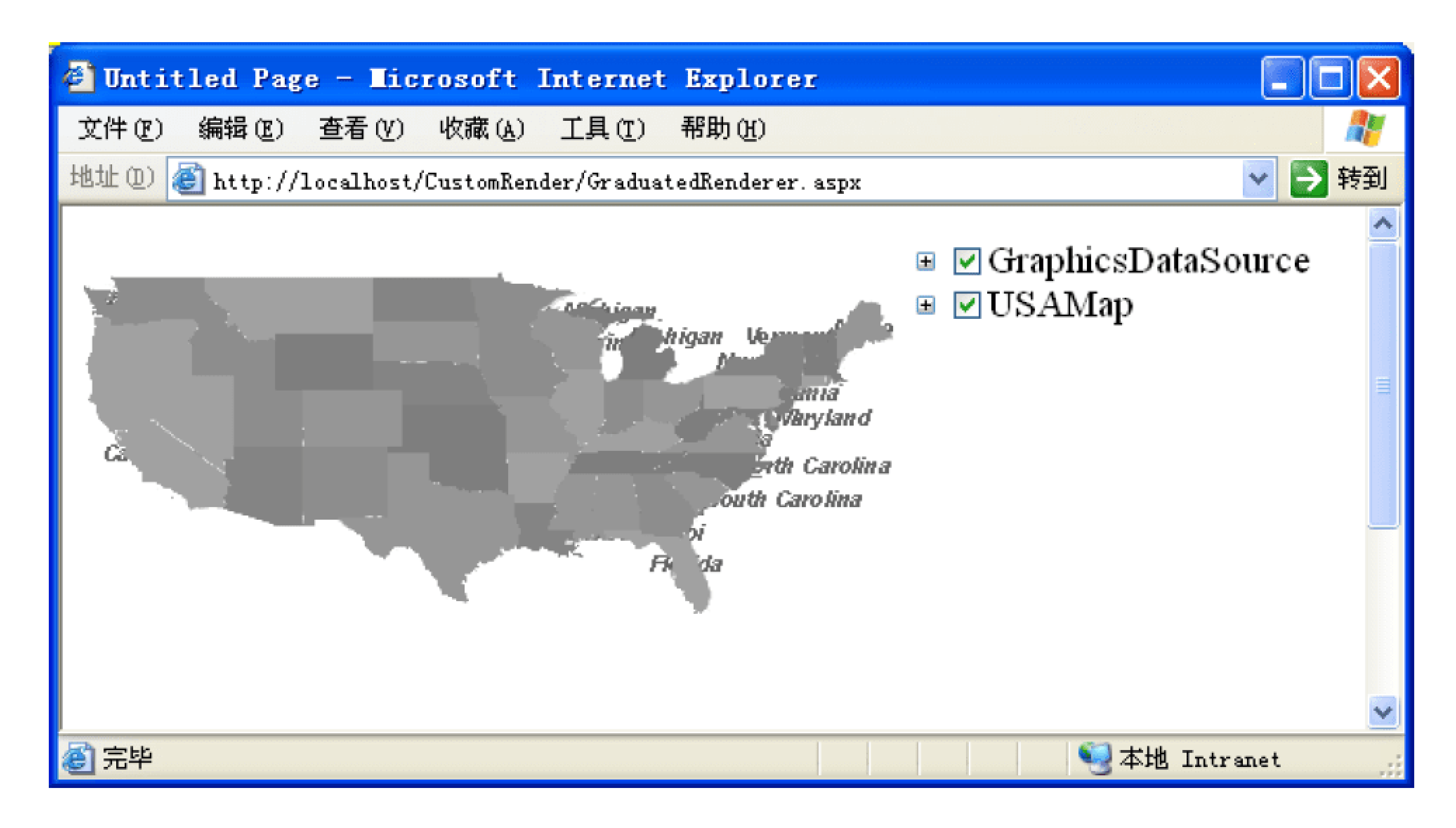


图 7.6 自定义范围专题图着色器

6. 自定义三维着色器

利用范围专题图着色器，可以大致看出各地区人口数量的分布情况，但是还不是太直观。最直观的方式是利用三维方式。本小节将带领读者分步骤完成一个三维着色器。

在 App_Code 文件夹中新加入一类，命名为 SimpleRenderer3D。在该类的文件头部加入如下命名空间的引用：

```
using ESRI.ArcGIS.ADF.Web.Display.Renderer;
using ESRI.ArcGIS.ADF.Web.Geometry;
using ESRI.ArcGIS.ADF.Web.Display.Symbol;
using System.Drawing.Drawing2D;
using System.Collections.Generic;
using System.Drawing;
```

将类的声明修改为如下代码，指定继承 RendererBase 类：

```
public class SimpleRenderer3D: RendererBase
```

然后加入表示填充颜色、边线颜色、透明度等的属性，代码如下：

```
// 填充颜色
private Color fillColor = Color.White;
public Color FillColor
{
    get
    {
        return fillColor;
    }
    set
    {
        fillColor = value;
    }
```

```
    }

    // 边线颜色
    private Color outlineColor = Color.Gray;
    public Color OutlineColor
    {
        get
        {
            return outlineColor;
        }
        set
        {
            outlineColor = value;
        }
    }

    // 透明程序
    private int transparency = 25;
    public int Transparency
    {
        get
        {
            return transparency;
        }
        set
        {
            transparency = value;
        }
    }

    // 环境
    private double ambience = 0.25;
    public double Ambience
    {
        get
        {
            return ambience;
        }
        set
        {
            if (ambience < 0 || ambience > 1)
            {
                throw new ArgumentOutOfRangeException("Ambience",
                                                      "环境值必须在 0 与 1 之间");
            }
            ambience = value;
        }
    }

    // 设置三维高度的字段
```

```
    private string heightColumnName;
    public string HeightColumnName
    {
        get
        {
            return heightColumnName;
        }
        set
        {
            heightColumnName = value;
        }
    }

    // 最大高度
    public double maxHeight = 50;
    public double MaxHeight
    {
        get
        {
            return maxHeight;
        }
        set
        {
            maxHeight = value;
        }
    }

    // 拉伸比例
    private double scaleHeight = 1.0;
    public double ScaleHeight
    {
        get
        {
            return scaleHeight;
        }
        set
        {
            scaleHeight = value;
        }
    }

    // 灯光方向
    private double lightDirection = 45 / 180 * Math.PI;
    public double LightDirection
    {
        get
        {
            return lightDirection / Math.PI * 180;
        }
        set
```

```
    {
        lightDirection = value / 180 * Math.PI;
    }
}
```

三维着色器主要实现了 IRender 接口的两个方法，它们的代码如下：

```
public override void GetAllSymbols(
System.Collections.Generic.List<FeatureSymbol> symbols)
{
    SimpleFillSymbol symbol = new SimpleFillSymbol(FillColor,
        OutlineColor, PolygonFillType.Solid);
    symbol.Transparency = Transparency;
    symbols.Add(symbol);
}

public override void Render(DataRow row, Graphics graphics,
DataColumn geometryColumn)
{
    if (row == null || graphics == null || geometryColumn == null)
        return;

    Geometry geometry = row[geometryColumn] as Geometry;
    if (geometry == null
        || geometry is ESRI.ArcGIS.ADF.Web.Geometry.Point)
        return;
    double height = MaxHeight;
    // 得到高度字段
    if (!string.IsNullOrEmpty(HeightColumnName)
        && row.Table.Columns.Contains(HeightColumnName))
    {
        double.TryParse(row[HeightColumnName].ToString(), out height);
    }
    height *= scaleHeight; //Multiply height with scale multiplier
    if (height > maxHeight)
        height = maxHeight; //Clip values beyong maxheight

    drawGraphic(geometry, graphics, height);
}
```

在上述代码中，主要调用 drawGraphic 方法来实现三维绘制要素的。该方法及其相关代码如下：

```
#region 三维绘制要素

private void drawGraphic(Geometry geometry, Graphics g, double height)
{
    if (geometry is ESRI.ArcGIS.ADF.Web.Geometry.Envelope) //Convert to polygon
    {
        Polygon polygon = new Polygon();
        polygon.Rings.Add(new Ring(geometry as Envelope));
        geometry = polygon;
```

```
        }
        if (geometry is Polygon)
        {
            drawPolygon(g, geometry as Polygon, height);
        }
    }

    private struct LineSegment : IComparable<LineSegment>
    {
        public ESRI.ArcGIS.ADF.Web.Geometry.Point Start;
        public ESRI.ArcGIS.ADF.Web.Geometry.Point End;
        public double Direction;

        public int CompareTo(LineSegment other)
        {
            return Math.Min(this.End.Y, this.Start.Y).CompareTo(
                    Math.Min(other.End.Y, other.Start.Y));
        }
    }

    // 将一点集合转换为连接这些点的线段列表
    private void getLineSegments(PointCollection points,
    ref List<LineSegment> list)
    {
        for (int i = 1; i < points.Count; i++)
        {
            LineSegment line = new LineSegment();
            line.Start = points[i - 1];
            line.End = points[i];
            line.Direction = getLineDirection(line.Start, line.End);
            list.Add(line);
        }
    }

    // 实现三维绘制要素的核心方法
    private void drawPolygon(Graphics g, Polygon polygon, double height)
    {
        foreach (Ring p in polygon.Rings)
        {
            List<LineSegment> list = new List<LineSegment>(p.Points.Count);
            getLineSegments(p.Points, ref list);
            foreach (Hole hole in p.Holes)
            {
                getLineSegments(hole.Points, ref list);
            }
            list.Sort(); // 进行排序
            // 第一步，绘制反面
            if (Transparency > 0) //ignore if not see-through
            {
                foreach (LineSegment line in list)
```

```
                {
                    if (Math.Abs(line.Direction) > HALF_PI)
                    {
                        drawLineSegment(g, line.Start, line.End, height,
                                         line.Direction, false);
                    }
                }
            }

            // 第二步，绘制前面
            foreach (LineSegment line in list)
            {
                if (Math.Abs(line.Direction) <= HALF PI)
                {
                    drawLineSegment(g, line.Start, line.End, height, line.Direction,
true);
                }
            }
        }
        Utility.FillPolygon(g, polygon, FillColor, Transparency, 0, (int)height);
    }

    private void drawLineSegment(Graphics g,
    ESRI.ArcGIS.ADF.Web.Geometry.Point start,
    ESRI.ArcGIS.ADF.Web.Geometry.Point end,
    double height, double direction, bool fill)
    {
        using (GraphicsPath path = new GraphicsPath())
        {
            path.AddPolygon(new System.Drawing.Point[] {
                  new System.Drawing.Point(
                        Convert.ToInt32(start.X),Convert.ToInt32(start.Y)),
                  new System.Drawing.Point(
                        Convert.ToInt32(end.X),Convert.ToInt32(end.Y)),
                  new System.Drawing.Point(
                        Convert.ToInt32(end.X),Convert.ToInt32(end.Y-height)),
                  new System.Drawing.Point(
                        Convert.ToInt32(start.X),Convert.ToInt32(start.Y-height)),
                  new System.Drawing.Point(
                        Convert.ToInt32(start.X),Convert.ToInt32(start.Y))
             });

            if (fill) //Fill facade
            {
                double b = calculateBrightness(direction + HALF PI);
                Color c = adjustBrightness(FillColor, b);
                Color fillColor = Color.FromArgb(
                                   Utility.TransparencyToAlpha(Transparency), c);
                using (SolidBrush brush = new SolidBrush(fillColor))
                {
```

```
                g.FillPath(brush, path);
                brush.Dispose();
            }
        }
        if (OutlineColor != Color.Transparent && OutlineColor != Color.Empty)
        {
            using (Pen pen = new Pen(OutlineColor, 0.5f))
            {
                g.DrawPath(pen, path);
                pen.Dispose();
            }
        }
        path.Dispose();
    }
}

#endregion

#region 辅助方法

private Color adjustBrightness(Color color, double brightnessFactor)
{
    int r = color.R;
    int g = color.G;
    int b = color.B;
    brightnessFactor *= (1 - ambience);
    r = Convert.ToInt32(r * (1 - brightnessFactor));
    g = Convert.ToInt32(g * (1 - brightnessFactor));
    b = Convert.ToInt32(b * (1 - brightnessFactor));
    if (r > 255) r = 255;
    else if (r < 0) r = 0;
    if (g > 255) g = 255;
    else if (g < 0) g = 0;
    if (b > 255) b = 255;
    else if (b < 0) b = 0;
    return Color.FromArgb(color.A, r, g, b);
}

private const double HALF PI = Math.PI * 0.5;
/// <summary>
/// 根据与光源的角度计算亮度系数
/// </summary>
/// <returns>0 表示垂直，1 表示同方向或反向</returns>
private double calculateBrightness(double angle)
{
    double diff = (lightDirection + HALF PI - angle);
    return Math.Abs(Math.Sin(-diff / 2));
}

private static double getLineDirection(
```

```
ESRI.ArcGIS.ADF.Web.Geometry.Point start,
ESRI.ArcGIS.ADF.Web.Geometry.Point end)
{
    double dx = end.X - start.X;
    double dy = end.Y - start.Y;
    return Math.Atan2(dy, dx);
}

#endregion
```

在工程新加入一 ASP.NET 页面，命名为 Renderer3DPage。

在 Renderer3DPage.aspx 页面中增加一地图资源管理器控件、一地图控件与一 Toc 控件。通过地图资源管理器控件的 MapResourcesItem 属性设置对话框，先加入一个 GraphicsLayer 类型的资源，命名为 GraphicsDataSource，然后加入 USAMap 地图资源。

在 Renderer3DPage 类的头部加入如下命名空间引用：

```
using ESRI.ArcGIS.ADF.Web.Display.Graphics;
using ESRI.ArcGIS.ADF.Web.Geometry;
using ESRI.ArcGIS.ADF.Web.DataSources.Graphics;
using ESRI.ArcGIS.ADF.Web.UI.WebControls;
using ESRI.ArcGIS.ADF.Web.DataSources;
```

然后在 Renderer3DPage 类中加入 Page 对象的 PreRender 事件处理方法，代码如下：

```
protected void Page_PreRender(object sender, EventArgs e)
{
    if (IsPostBack)
        return;

    MapResourceItem resourceItem =
         MapResourceManager1.ResourceItems.Find("GraphicsDataSource");
    MapResource resource = resourceItem.Resource as MapResource;
    if (resource != null)
    {
        if (Map1.Extent == null)
        {
            IMapResource priResource = Map1.PrimaryMapResourceInstance;
        }

        FeatureGraphicsLayer layer =
            GenerateGraphicsHelper.AttributeQuery(Map1, 50);
        addLayer(resource, layer);
    }
    Map1.RefreshResource("GraphicsDataSource");
}

private static void addLayer(MapResource resource, FeatureGraphicsLayer layer)
{
    if (layer != null)
    {
```

```
            //Apply renderer
            SimpleRenderer3D renderer = new SimpleRenderer3D();
            renderer.FillColor = System.Drawing.Color.White;
            renderer.HeightColumnName = "POP1999";
            renderer.ScaleHeight = 1;
            renderer.MaxHeight = 50;
            layer.Renderer = renderer;

            if (resource.Graphics.Tables.Contains(layer.TableName))
                resource.Graphics.Tables.Remove(layer.TableName);
            resource.Graphics.Tables.Add(layer);
        }
    }
```

编译并运行程序，可得到如图 7.7 所示的以三维形状表示的要素，其中高度是根据人口的多少来设置的。从图中可以直观地看出美国人口分布情况。

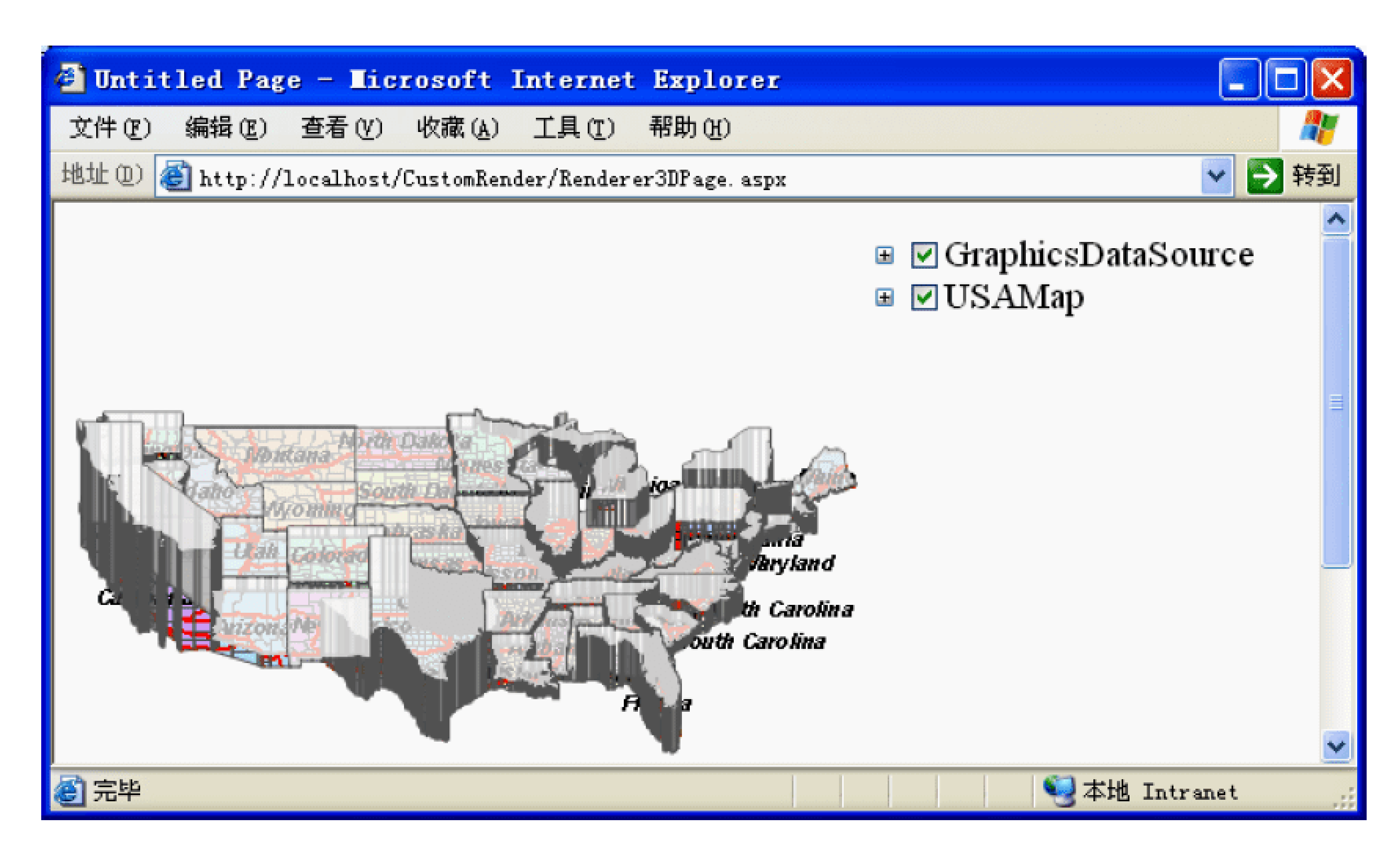


图 7.7 自定义三维着色器

7.4 在 GIS 服务器端操作图形

在 4.4.3 节中介绍了如何通过设置 MapDescription 对象的 CustomGraphics 属性，从而实现在 GIS 服务器端高亮显示选择的要素。在 5.5.3 节中仅介绍了如何获取服务器上下文以及服务器对象，从而更细粒度地来控制服务器的地图。

在本节中我们将通过绘制专题图的实例，进一步介绍如何在服务器端操作图形。

服务器端设置专题图用到的主要程序集是 ESRI.ArcGIS.Carto。

在 Visual Studio 2005 中，利用 File 菜单的 New Web Site 创建一个新的站点，命名为 ThemeMap。

在 Default.aspx 页面中增加一地图资源管理器控件、一地图控件与一 Toc 控件。通过地图资源管理器控件的 MapResourcesItem 属性设置对话框，先加入一个 GraphicsLayer 类型的资源，命名为 GraphicsDataSource，然后加入 USAMap 地图资源。

通过工程的右键菜单的 Add ArcGIS Identity 命令，加入连接 USAMap 地图资源的身份信息。

再在 Default.aspx 页面中增加一 HTML 类型的 Select 控件，通过直接修改代码，在该控件中加入一些选项与设置 onchange 事件的处理函数。代码如下：

```
<select id="Select1" onchange="SetTheme(this)"
 style="left: 416px; width: 152px; position: absolute; top: 331px">
   <option value="1" >独立值专题图</option>
   <option value="2" >范围专题图</option>
   <option value="3" >柱状专题图</option>
   <option value="4" >饼状专题图</option>
   <option value="5" >等级符号专题图</option>
   <option value="6" >点密度专题图</option>
</select>
```

在<head>与</head>之间加入 SetTheme 函数的代码，如下所示：

```
<script  language="javascript" type="text/javascript">
   function SetTheme(selectCtrl)
   {
      var message = selectCtrl.value;
      var context = 'Page1';
      <%=setThemCallBack%>
   }
</script>
```

上述代码执行时调用 setThemCallBack 代表的回调代码。

在 Default.aspx.cs 文件中，首先将类的声明代码行后面加入对 ICallbackEventHandler 接口的实现，然后在类中加入如下两个字段：

```
public string setThemCallBack; // 客户端函数的引用
private string callbackArg; // 回调事件的结果
```

在 Page_Load 方法中加入如下代码，获取一个对客户端函数的引用：

```
protected void Page Load(object sender, EventArgs e)
{
   setThemCallBack = ClientScript.GetCallbackEventReference(this,
      "message", "processCallbackResult", "context", "postBackError", true);
}
```

加入 ICallbackEventHandler 接口要求的两个方法的实现代码，如下所示：

```
void ICallbackEventHandler.RaiseCallbackEvent(string eventArgument) {
   callbackArg = eventArgument;
}

string ICallbackEventHandler.GetCallbackResult() {
   string[] fields = new string[2];
   fields[0] = "POP1990";
   fields[1] = "POP1999";

   if (callbackArg == "1") {
   }
```

```
        else if(callbackArg == "2") {
        }
        else if (callbackArg == "3") {
        }
    else if(callbackArg == "4") {
        }
        else if (callbackArg == "5") {
        }
        else if (callbackArg == "6") {
        }

        return "";
    }
```

通过上述的代码，整个应用程序的框架就搭好了，下面需要做的就是针对每个类型的专题图，编写实现代码。

7.4.1 独立值专题图

独立值专题图是一种比较简单的专题地图。它使用不同的颜色、符号或线形来显示不同的数据。根据独立值绘制地图对象的专题地图有助于强调数据的类型差异而不是显示定量信息(如给定区域内的商店类型、分区类型等等)。因此，当用户只需要使用单一的数据值来渲染时，可以使用独立值专题图。

例如，要制作一个区域的土地利用类型专题图，该区域可能的土地利用类型有工业用地、农业用地和居民区用地等，这时可以利用打开表中的土地利用类型来制作独立值专题图，不同的利用类型赋予不同的颜色，从而产生简单明了的土地利用专题图。

ESRI.ArcGIS.Carto 中的 IUniqueValueRenderer 接口用于创建独立值专题图。

该接口管理一组类别与符号。数据中的唯一值可用于定义一类别，用单独的一种符号来表示。该数据的值可以是一组值的组合，这样可以用一个符号来表示不同的值。要实现这种操作，需要使用 AddReferenceValue 方法与 ReferenceValue 属性。

在工程中加入 ASP.NET 文件夹 App_Code，在其中新增加 GisFunctionality 类。在该类中首先加入如下一些命名空间的引用：

```
    using ESRI.ArcGIS.Display;
    using ESRI.ArcGIS.ADF.Web.DataSources.ArcGISServer;
    using ESRI.ArcGIS.ADF.Connection.AGS;
    using ESRI.ArcGIS.Server;
    using ESRI.ArcGIS.Carto;
    using ESRI.ArcGIS.Geodatabase;
    using ESRI.ArcGIS.ADF.Web.UI.WebControls;
```

我们利用名为 CreateUniqueValueRenderer 的静态方法来实现独立值专题图。首先在 GisFunctionality 类中增加上述方法的声明，代码如下：

```
    public static string CreateUniqueValueRenderer(
    ESRI.ArcGIS.ADF.Web.UI.WebControls.Map map,
```

```
    Toc toc, int layerID, string field)
    {
    }
```

在该方法中，参数 map 与 toc 分别代表页面中的地图控件与 Toc 控件，layerID 是需要设置专题图图层在地图资源中的位置序号，而 field 参数设置的是使用哪个字段来设置独立值。

在该方法中，首先需要完成的就是从地图资源中得到服务器上下文。代码如下：

```
    // 得到指定名称的资源的地图功能
    MapFunctionality gisfunc = map.GetFunctionality("USAMap")
                                    as MapFunctionality;
    if (gisfunc == null)
        return "";

    MapDescription mapDesc = gisfunc.MapDescription as MapDescription;
    MapResourceLocal mapResLocal = gisfunc.MapResource as MapResourceLocal;
    // 得到服务器上下文对象
    IServerContext serverContext = mapResLocal.ServerContextInfo.ServerContext;
    if (serverContext == null)
        return "";
```

接下来需要得到的就是指定索引的图层，代码如下：

```
    MapServer mapServer = mapResLocal.MapServer as MapServer;
    IMapServerObjects mapServerObj = mapServer as IMapServerObjects;
    string mapName = mapServer.get MapName(0);
    ILayer layer = mapServerObj.get Layer(mapName, layerID);
    IGeoFeatureLayer featureLyer = layer as IGeoFeatureLayer;
```

然后就可以设置该图层的着色器对象了。需要先创建该着色器对象。创建代码如下：

```
    IUniqueValueRenderer render = serverContext.CreateObject(
    "esriCarto.UniqueValueRenderer") as IUniqueValueRenderer;
```

读者应该注意的是，由于程序是在 Web 端运行，因此不能用 new 操作来在 GIS 服务器上创建对象，这样创建的对象仍然是本地对象，而非服务器对象。因此需要调用服务器上下文对象的 CreateObject 方法来创建。

专题图创建的大部分工作，就是设置着色器对象的属性。通过如下代码设置着色器的一般与默认属性值：

```
    ISimpleFillSymbol symbol = serverContext.CreateObject(
                "esriDisplay.SimpleFillSymbol") as SimpleFillSymbol;
    symbol.Color = GetRGB(serverContext, 239, 228, 249);
    symbol.Outline.Width = 0.4;
    render.FieldCount = 1;
    render.set Field(0, field);
    render.DefaultSymbol = symbol as ISymbol;
```

下面要实现的就是针对指定字段中不同的值设置单独的不同的符号。因此需要对图层中的属性进行循环，可以通过游标对象来进行循环。加入如下代码，得到循环游标：

```
    ICursor pCursor = featureLyer.Search(null, true) as ICursor;
```

```
IRow row = pCursor.NextRow();
```

上述代码通过游标对象的 NextRow 方法，将游标位置指向图层中的第一条记录。

由于传入的参数指明的是字段名称，而通过获取游标只能用字段索引得到字段的值。因此需要在循环之前，得到字段的索引。代码如下：

```
int fieldIndex = 0;
for (int j = 0; j < row.Fields.FieldCount; j++) {
    string fieldName = row.Fields.get_Field(j).Name.ToUpperInvariant();
    if (fieldName == field) {
        fieldIndex = j;
        break;
    }
}
```

下面就可以循环设置着色器对象的 Value 属性了。循环代码如下：

```
Random ranColor = new Random();
// 循环设置每个值的符号
while (row != null) {
    string fieldValue = row.get_Value(fieldIndex).ToString();

    // 先判断该值是否已经存在于着色器中
    bool hasFound = false;
    for (int i = 0; i < render.ValueCount; i++) {
        if (render.get_Value(i) == fieldValue) {
            hasFound = true;
            break;
        }
    }

    // 只加入没有存在的
    if (hasFound == false) {
        IFillSymbol fillSymbol =
              serverContext.CreateObject("esriDisplay.SimpleFillSymbol")
              as IFillSymbol;
        fillSymbol.Color =
              GetRGB(serverContext, ranColor.Next(255),
              ranColor.Next(255), ranColor.Next(255));
        render.AddValue(fieldValue, "Name", fillSymbol as ISymbol);
        render.set_Label(fieldValue, fieldValue);
        render.set_Symbol(fieldValue, fillSymbol as ISymbol);
    }

    row = pCursor.NextRow();
}
```

在上面的循环中，由于需要设置的不同的值才设置不同的符号，因此需要先判断本记录该字段的值已经加入到着色器中了。只有没有加入着色器的值，才需要设置一个新的序号。

在设置着色器符号与注记的代码中，利用 GetRGB 方法从随机的颜色值中得到填充符号对象的颜色，然后利用着色器对象的 AddValue 方法，新增加独立值符号，用 set_Label 方法设置注记。

最后要实现的是将着色器应用到图层对象上，并刷新地图与 Toc。代码如下：

```
// 应用专题到指定图层
featureLyer.Renderer = render as IFeatureRenderer;
// 释放服务器上下文对象
serverContext.ReleaseContext();

// 刷新地图及 Toc
toc.Refresh();
CallbackResult tocCallbackResult = RefreshControlHtml(toc);
map.CallbackResults.Add(tocCallbackResult);
map.Refresh();
return map.CallbackResults.ToString();
```

在上述代码中用到了 GetRGB 与 RefreshControlHtml 两个辅助方法，它们的代码如下：

```
private static IColor GetRGB(
IServerContext serverContext, int red, int green, int blue) {
    IRgbColor rgbColor = serverContext.CreateObject(
                       "esriDisplay.RGBColor") as IRgbColor;
    IColor color = rgbColor as IColor;
    rgbColor.Red = red;
    rgbColor.Green = green;
    rgbColor.Blue = blue;
    return color;
}

private static CallbackResult RefreshControlHtml(Control control) {
    System.IO.StringWriter stringWriter = new System.IO.StringWriter();
    HtmlTextWriter writer = new HtmlTextWriter(stringWriter);
    control.RenderControl(writer);
    string htmlContent = stringWriter.ToString();
    stringWriter.Close();
    return new CallbackResult(control, "content", htmlContent);
}
```

切换到 Default.aspx.cs 文件中，找到 GetCallbackResult 方法，在 callbackArg 为 1 的部分，加入对 CreateUniqueValueRenderer 方法的调用。代码如下：

```
    return GisFunctionality.CreateUniqueValueRenderer(Map1, Toc1, 1,
"STATE_NAME");
```

上述代码表示，针对 USAMap 地图资源中的第二个图层（即美国州行政区划）的州名设置独立值专题图，即每个州使用一个不同符号显示。

编译并运行程序，程序运行效果如图 7.8 所示。

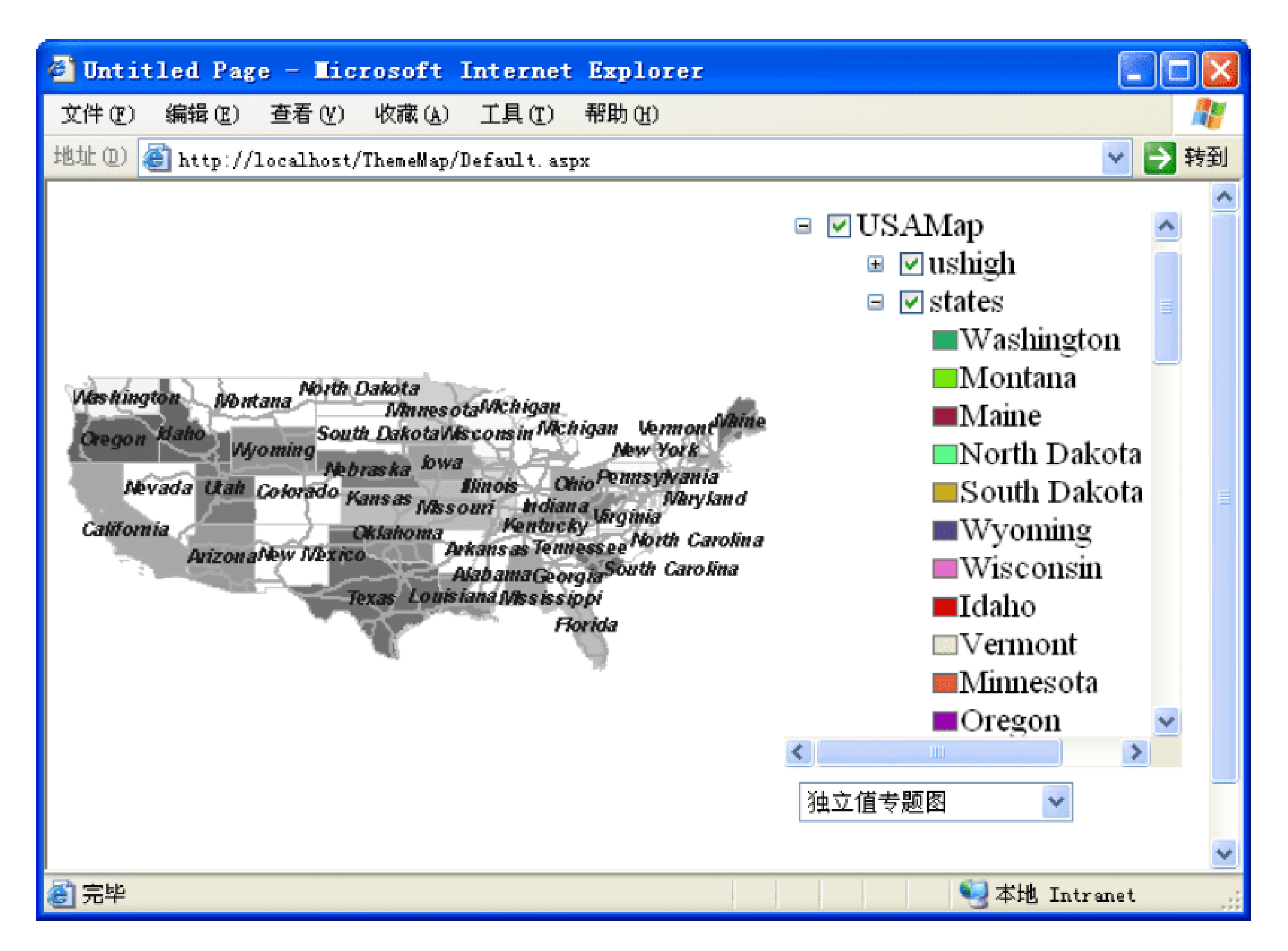


图 7.8 独立值专题图

7.4.2 范围专题图

范围专题图是按照设置的范围显示数据，这些范围用颜色和图案进行渲染。范围专题图能够通过点、线和区域来说明数值，在反映数值和地理区域的关系（如销售数字、家庭收入），或显示比率信息如人口密度（人口除以面积）时很有用。

实现范围专地图最重要的工作就是进行范围的划分。ArcGIS 使用 5 个范围划分方法自动创建范围专题图，它们分别是等计数（Equal Count）、等范围（Equal Ranges）、自然划分（Natural Break）、标准差（Standard Deviation）和分位数（Quantile）。

我们利用名为 GreateClassBreaksRender 的静态方法来实现范围专题图。首先在 GisFunctionality 类中增加上述方法的声明，代码如下：

```
public static string CreateClassBreaksRenderer(
ESRI.ArcGIS.ADF.Web.UI.WebControls.Map map,Toc toc, int layerID,
string field, int classCount) {
}
```

在该方法中，参数 map 与 toc 分别代表页面中的地图控件与 Toc 控件，layerID 是需要设置专题图图层在地图资源中的位置序号，而 field 参数设置的是使用哪个字段来统计。

在该方法中，首先需要完成的就是从地图资源中得到服务器上下文，并从中得到图层对象。代码如下：

```
MapFunctionality gisfunc = map.GetFunctionality("USAMap")
                                    as MapFunctionality;
if (gisfunc == null)
    return "";
```

```
MapDescription mapDesc = gisfunc.MapDescription as MapDescription;
MapResourceLocal mapResLocal = gisfunc.MapResource as MapResourceLocal;
IServerContext serverContext = mapResLocal.ServerContextInfo.ServerContext;
if (serverContext == null)
    return "";

MapServer mapServer = mapResLocal.MapServer as MapServer;
IMapServerObjects mapServerObj = mapServer as IMapServerObjects;
string mapName = mapServer.get_MapName(0);
ILayer layer = mapServerObj.get_Layer(mapName, layerID);
IGeoFeatureLayer featureLyer = layer as IGeoFeatureLayer;
```

对于范围专题图，由于需要对数据进行划分，那么首先需要对这些数据进行统计。加入如下代码：

```
IBasicHistogram pBasicHist = null;
pBasicHist = serverContext.CreateObject("esriCarto.BasicTableHistogram")
                as IBasicHistogram;
ITableHistogram pTableHist = pBasicHist as ITableHistogram;
pTableHist.Field = field;
pTableHist.Table = featureLyer.FeatureClass as ITable;
object xVals = null;
object frqs;
pBasicHist.GetHistogram(out xVals, out frqs);
```

上面的代码，对 field 指定的字段中的值进行统计，不同的值记录在 xVals 变量中，每个值对应的个数记录在 frqs 中。

下来要完成的是针对这些统计数据，进行指定方法的划分，我们这里使用分位数的方式划分。代码如下：

```
IClassifyGEN quantile = serverContext.CreateObject("esriSystem.Quantile")
                          as IClassifyGEN;
quantile.Classify(xVals, frqs, ref classCount);
System.Double[] classBreaks = quantile.ClassBreaks as System.Double[];
```

通过上述代码后，数据分类放在 classBreaks 数组中了。

下面通过一个算法自动根据这些数据以及起始颜色与终止颜色，自动插入其他颜色的值。代码如下：

```
IAlgorithmicColorRamp colorRamp = null;
colorRamp = serverContext.CreateObject("esriDisplay.AlgorithmicColorRamp")
               as IAlgorithmicColorRamp;
colorRamp.Algorithm = esriColorRampAlgorithm.esriCIELabAlgorithm;
colorRamp.FromColor = GetRGB(serverContext, 255, 210, 210);
colorRamp.ToColor = GetRGB(serverContext, 190, 0, 170);
colorRamp.Size = classCount;
bool ok = true;
colorRamp.CreateRamp(out ok);
```

下面就可以创建着色器对象了。代码如下：

```
    IClassBreaksRenderer render = null;
    render = serverContext.CreateObject("esriCarto.ClassBreaksRenderer")
                as IClassBreaksRenderer;
    render.Field = field;
    render.BreakCount = classCount;
    render.MinimumBreak = classBreaks[0];

    IEnumColors pEnumColors = colorRamp.Colors;
    for (int i = 0; i < classCount; i++) {
        render.set_Break(i, classBreaks[i + 1]);
        render.set_Label(i, classBreaks[i] + "--" + classBreaks[i + 1]);
        IFillSymbol                         fillSymbol                         =
serverContext.CreateObject("esriDisplay.SimpleFillSymbol")
            as IFillSymbol;
        fillSymbol.Color = pEnumColors.Next();
        render.set_Symbol(i, fillSymbol as ISymbol);
    }
```

最后需要加入的代码是将着色器应用到图层上，并刷新地图与 Toc。代码如下：

```
    // 应用专题到指定图层
    featureLyer.Renderer = render as IFeatureRenderer;
    // 释放服务器上下文对象
    serverContext.ReleaseContext();

    // 刷新地图及 Toc
    toc.Refresh();
    CallbackResult tocCallbackResult = RefreshControlHtml(toc);
    map.CallbackResults.Add(tocCallbackResult);
    map.Refresh();
    return map.CallbackResults.ToString();
```

切换到 Default.aspx.cs 文件中，找到 GetCallbackResult 方法，在 callbackArg 为 2 的部分，加入对 CreateUniqueValueRenderer 方法的调用。代码如下：

```
    return GisFunctionality.CreateClassBreaksRenderer(Map1, Toc1, 1, "POP1999" ,
5);
```

上述代码表示，针对 USAMap 地图资源中的第二个图层（即美国州行政区划）的各州 1999 年人口数据设置范围值专题图，划分为 5 类。

编译并运行程序，程序运行效果如图 7.9 所示。

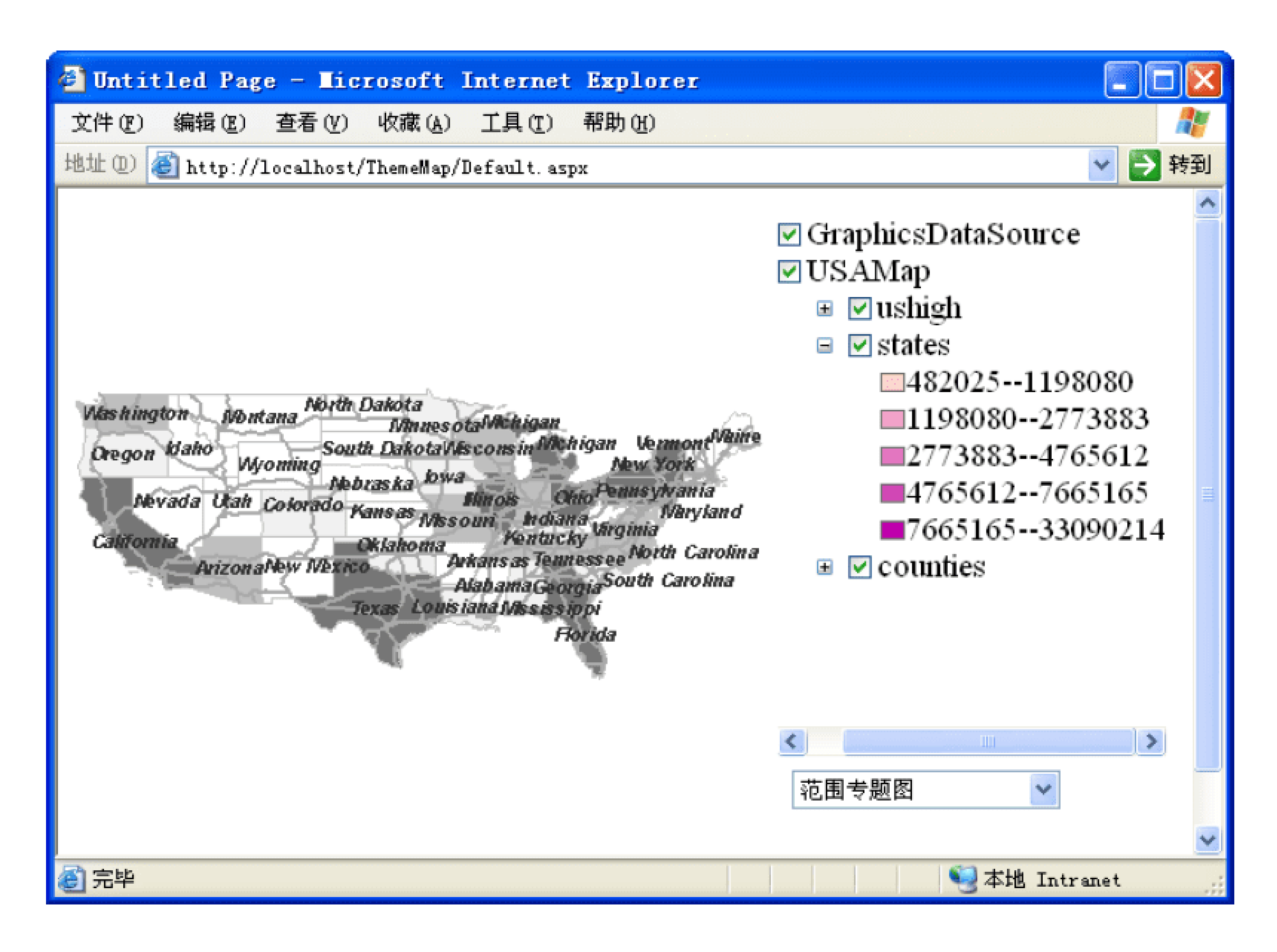


图 7.9　范围专题图

7.4.3　柱状专题图

柱状专题图是将表中每条记录的专题变量显示为一个柱状图。使用柱状图可分析地图中每条记录的多个变量。比较每个柱状图中各直方条的大小可考察表中某条记录，而比较所有柱状图中某一条直方的大小可考察所有记录的某个变量，比较各柱状图的高度则可考察整张表。用柱状图来表达负值时，该条会沿柱状图反方向伸展。在叠加柱状图中不显示负值。

例如，要创建一个关于人均收入的柱状专题图，假设有一张中国各个省份的表，其中包括 1985 年和 1995 年的人均收入，通过在每个省显示具有两个直方条的统计图，一个代表 1985 年的人均收入，一个代表 1995 年的人均收入。可以比较 1985 年和 1995 年每个省的人均收入增长情况，也可以分析几个不同省之间人均收入或人均收入的增长情况。

注意，虽然柱状图可以表示多个变量，但是为了得到最佳显示效果，在分析中每个柱状图最好不要超过 6 个直方条。

利用名为 CreateBarRenderer 的静态方法来实现柱状专题图。首先在 GisFunctionality 类中增加上述方法的声明，代码如下：

```
public static string CreateBarRenderer (
ESRI.ArcGIS.ADF.Web.UI.WebControls.Map map,Toc toc, int layerID,
string[] fields)
{
}
```

同样，在该方法中，参数 map 与 toc 分别代表页面中的地图控件与 Toc 控件，layerID 是需要设置专题图图层在地图资源中的位置序号，而 fields 参数设置的是使用哪些字段来统计。

在该方法中，完成从地图资源中得到服务器上下文，并从中得到图层对象。代码如下：

```
    MapFunctionality    gisfunc    =    map.GetFunctionality("USAMap")    as
MapFunctionality;
    if (gisfunc == null)
        return "";

    MapDescription mapDesc = gisfunc.MapDescription as MapDescription;
    MapResourceLocal mapResLocal = gisfunc.MapResource as MapResourceLocal;
    IServerContext serverContext = mapResLocal.ServerContextInfo.ServerContext;
    if (serverContext == null)
        return "";

    MapServer mapServer = mapResLocal.MapServer as MapServer;
    IMapServerObjects mapServerObj = mapServer as IMapServerObjects;
    string mapName = mapServer.get MapName(0);
    ILayer layer = mapServerObj.get Layer(mapName, layerID);
    IGeoFeatureLayer featureLyer = layer as IGeoFeatureLayer;
```

接着设置专题图元素的属性名称列表，代码如下：

```
    IChartRenderer chartRender = null;
    chartRender = serverContext.CreateObject("esriCarto.ChartRenderer")
        as IChartRenderer;
    IRendererFields renderFields = chartRender as IRendererFields;
    foreach (string var in fields)
    {
        renderFields.AddField(var, var);
    }
```

接着实例化图表对象并取得元素指定属性的最大值，代码如下：

```
    IBarChartSymbol barChartSymbol = null;
    barChartSymbol = serverContext.CreateObject("esriDisplay.BarChartSymbol")
        as IBarChartSymbol;
    IChartSymbol chartSymbol = barChartSymbol as IChartSymbol;
    chartSymbol.MaxValue = GetStaMaxMin(featureLyer, fields)[0];
    barChartSymbol.Width = 8;
    IMarkerSymbol markerSymbol = barChartSymbol as IMarkerSymbol;
    markerSymbol.Size = 50;
```

接着加入设置柱状图每列填充效果的代码，如下所示：

```
    ISymbolArray symbolArray = barChartSymbol as ISymbolArray;
    Random ranColor = new Random();
    for (int i = 0; i < fields.Length; i++)
    {
        IFillSymbol fillSymbol =
            serverContext.CreateObject("esriDisplay.SimpleFillSymbol")
            as IFillSymbol;
        fillSymbol.Color = GetRGB(serverContext, ranColor.Next(255),
                                   ranColor.Next(255), ranColor.Next(255));
        symbolArray.AddSymbol((ISymbol)fillSymbol);
    }
```

然后设置图层背景，代码如下：

```
ISimpleFillSymbol symbol =
    serverContext.CreateObject("esriDisplay.SimpleFillSymbol")
    as SimpleFillSymbol;
symbol.Color = GetRGB(serverContext, 239, 228, 249);
chartRender.BaseSymbol = symbol as ISymbol;
```

并将柱状专题应用到指定图层，代码如下：

```
chartRender.ChartSymbol = barChartSymbol as IChartSymbol;
chartRender.Label = "Test";
chartRender.UseOverposter = false;
chartRender.CreateLegend();
featureLyer.Renderer = chartRender as IFeatureRenderer;
```

最后释放服务器上下文对象，并刷新地图与 Toc。代码如下：

```
serverContext.ReleaseContext();
map.Refresh();
toc.Refresh();
CallbackResult tocCallbackResult = RefreshControlHtml(toc);
map.CallbackResults.Add(tocCallbackResult);
return map.CallbackResults.ToString();
```

上述代码利用 GetStaMaxMin 方法来得到某图层中某字段的最大值与最小值。代码如下：

```
private static double[] GetStaMaxMin(IGeoFeatureLayer featureLyer,
string[] fields) {
    double pMaxValue = 0;
    double pMinValue = 0;
    double pStaMax;
    double pStaMin;
    double[] PMaxMin = new double[2];
    for (int i = 0; i < fields.Length; i++) {
        ICursor pCursor = featureLyer.Search(null, true) as ICursor;
        IDataStatistics pDataSta = new DataStatisticsClass();
        pDataSta.Cursor = pCursor;
        pDataSta.Field = fields[i];
        pStaMax = pDataSta.Statistics.Maximum;
        pStaMin = pDataSta.Statistics.Minimum;
        if (pMaxValue < pStaMax) {
            pMaxValue = pStaMax;
        }
        if (pMinValue > pStaMin) {
            pMinValue = pStaMin;
        }
    }
    PMaxMin[0] = pMaxValue;
    PMaxMin[1] = pMinValue;
    return PMaxMin;
}
```

切换到 Default.aspx.cs 文件中，找到 GetCallbackResult 方法，在 callbackArg 为 3 的部分，加入对 CreateUniqueValueRenderer 方法的调用。代码如下：

```
return GisFunctionality.CreateBarRenderer(Map1, Toc1, 1, fields);
```

上述代码表示，针对 USAMap 地图资源中的第二个图层（即美国州行政区划）的各州 1990 年与 1999 年人口数据设置柱状专题图。

编译并运行程序，程序运行效果如图 7.10 所示。

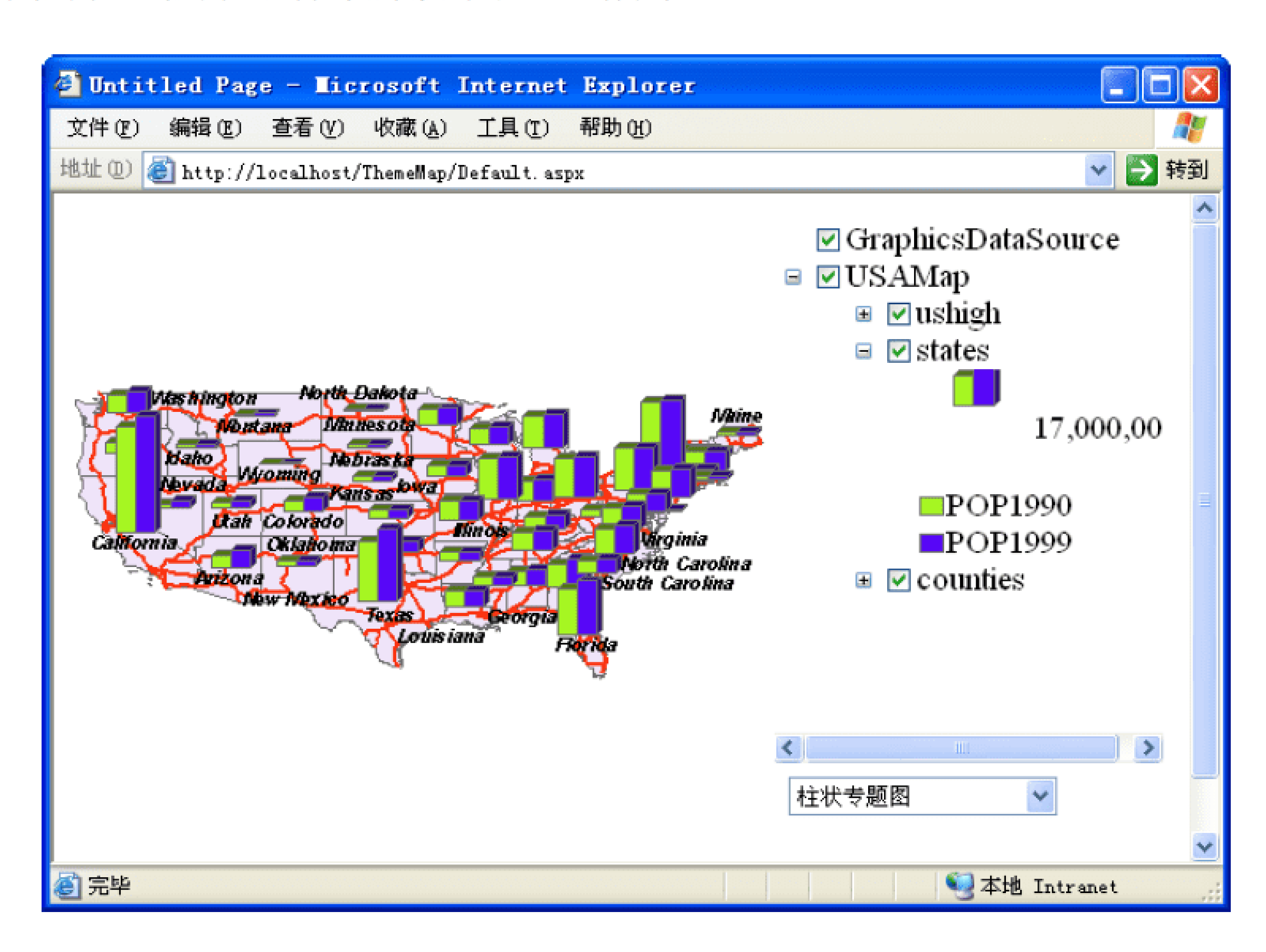


图 7.10 柱状专题图

7.4.4 饼状专题图

也可以以饼图显示表中各记录的专题变量。在地图上使用饼图同样可一次分析多个变量，比较每个图中饼扇的大小可考察表中某条记录，比较所有饼图中某一个饼扇，可考察所有记录中某个变量的变化，比较各饼图的直径则可考察整张表。

饼图与柱状图类似，饼图允许每次对记录分析多个变量，但柱状图比较的是直方条的高度，而饼图比较的饼中的饼扇，或是对所有饼分析某个特定的饼扇。

饼图与柱状图对于分析人口统计数据都是非常有用的。例如，假设有中国人口统计信息表，表中包含了各个省的人口总数以及几个主要人口群体（男、女、城市人口及农村人口等）组成。通过创建饼图专题图，可以显示每个人口群体的人数及其在每个饼中所占的份额，这样就可以查看不同省的人口总数及其人口群体是如何变化的。与创建柱状图一样，为获取最佳效果，在分析时每张饼图最好不要超过 6 个饼扇。

我们利用名为 CreateBarRenderer 的静态方法来实现饼状专题图。首先在 GisFunctionality 类中增加上述方法的声明，代码如下：

```
public static string CreateBarRenderer (
ESRI.ArcGIS.ADF.Web.UI.WebControls.Map map,Toc toc, int layerID,
```

```
    string[] fields)
    {
    }
```

参数 map 与 toc 分别代表页面中的地图控件与 Toc 控件，layerID 是需要设置专题图图层在地图资源中的位置序号，而 fields 参数设置的是使用哪些字段来统计。

完成从地图资源中得到服务器上下文，并从中得到图层对象。代码如下：

```
MapFunctionality gisfunc = map.GetFunctionality("USAMap")
as MapFunctionality;
if (gisfunc == null)
    return "";

MapDescription mapDesc = gisfunc.MapDescription as MapDescription;
MapResourceLocal mapResLocal = gisfunc.MapResource as MapResourceLocal;
IServerContext serverContext = mapResLocal.ServerContextInfo.ServerContext;
if (serverContext == null)
    return "";

MapServer mapServer = mapResLocal.MapServer as MapServer;
IMapServerObjects mapServerObj = mapServer as IMapServerObjects;
string mapName = mapServer.get MapName(0);
ILayer layer = mapServerObj.get Layer(mapName, layerID);
IGeoFeatureLayer featureLyer = layer as IGeoFeatureLayer;
```

接着设置专题图元素的属性名称列表，代码如下：

```
IChartRenderer chartRender = null;
chartRender = serverContext.CreateObject("esriCarto.ChartRenderer")
    as IChartRenderer;
IRendererFields renderFields = chartRender as IRendererFields;
foreach (string var in fields)
{
    renderFields.AddField(var, var);
}
```

接着实例化图表对象并取得元素指定属性的最大值，代码如下：

```
IPieChartSymbol pieChartSymbol = null;
pieChartSymbol = serverContext.CreateObject("esriDisplay.PieChartSymbol")
    as IPieChartSymbol;
IChartSymbol chartSymbol = pieChartSymbol as IChartSymbol;
pieChartSymbol.Clockwise = true;
pieChartSymbol.UseOutline = true;
chartSymbol.MaxValue = GetStaMaxMin(featureLyer, fields)[0];
```

设置饼图外围线的代码如下：

```
ISimpleLineSymbol outLine = null;
outLine = serverContext.CreateObject("esriDisplay.SimpleLineSymbol")
    as ISimpleLineSymbol;
outLine.Color = GetRGB(serverContext, 255, 0, 255);
```

```
outLine.Style = esriSimpleLineStyle.esriSLSSolid;
outLine.Width = 1;
pieChartSymbol.Outline = outLine;
IMarkerSymbol markerSymbol = pieChartSymbol as IMarkerSymbol;
markerSymbol.Size = 25;
```

设置饼状图外围填充效果 代码如下：

```
ISymbolArray symbolArray = pieChartSymbol as ISymbolArray;
ISimpleFillSymbol symbol = null;
symbol = serverContext.CreateObject("esriDisplay.SimpleFillSymbol")
    as SimpleFillSymbol;
symbol.Color = GetRGB(serverContext, 213, 212, 252);
symbol.Outline = outLine;
```

通过循环，设置饼状图每列填充效果。代码如下：

```
Random ranColor = new Random();
for (int i = 0; i < fields.Length; i++)
{
    IFillSymbol fillSymbol =
       serverContext.CreateObject("esriDisplay.SimpleFillSymbol")
       as IFillSymbol;
    fillSymbol.Color = GetRGB(serverContext, ranColor.Next(255),
                              ranColor.Next(255), ranColor.Next(255));
    symbolArray.AddSymbol((ISymbol)fillSymbol);
}
```

接着设置地图图层背景，代码如下：

```
symbol = serverContext.CreateObject("esriDisplay.SimpleFillSymbol")
          as SimpleFillSymbol;
symbol.Color = GetRGB(serverContext, 239, 228, 249);
chartRender.BaseSymbol = symbol as ISymbol;
```

然后设置饼状图表属性，代码如下：

```
IPieChartRenderer pieChartRenderer = chartRender as IPieChartRenderer;
pieChartRenderer.MinValue = 453588;
pieChartRenderer.MinSize = 10;
pieChartRenderer.FlanneryCompensation = false;
pieChartRenderer.ProportionalBySum = true;
pieChartRenderer.ProportionalField = fields[0];
chartRender.ChartSymbol = pieChartSymbol as IChartSymbol;
chartRender.Label = "Population";
```

最后将饼状专题应用到指定图层，并刷新地图与 Toc。代码如下：

```
chartRender.UseOverposter = false;
chartRender.CreateLegend();
featureLyer.Renderer = chartRender as IFeatureRenderer;
// 释放服务器上下文对象
serverContext.ReleaseContext();
toc.Refresh();
```

```
    CallbackResult tocCallbackResult = RefreshControlHtml(toc);
    map.CallbackResults.Add(tocCallbackResult);
    map.Refresh();
    return map.CallbackResults.ToString();
```

切换到 Default.aspx.cs 文件中，找到 GetCallbackResult 方法，在 callbackArg 为 4 的部分，加入对 CreateUniqueValueRenderer 方法的调用。代码如下：

```
    return GisFunctionality.CreatePieRenderer(Map1, Toc1, 1, fields);
```

上述代码表示，针对 USAMap 地图资源中的第二个图层（即美国州行政区划）的各州 1990 年与 1999 年人口数据设置饼图专题图。

编译并运行程序，程序运行效果如图 7.11 所示。

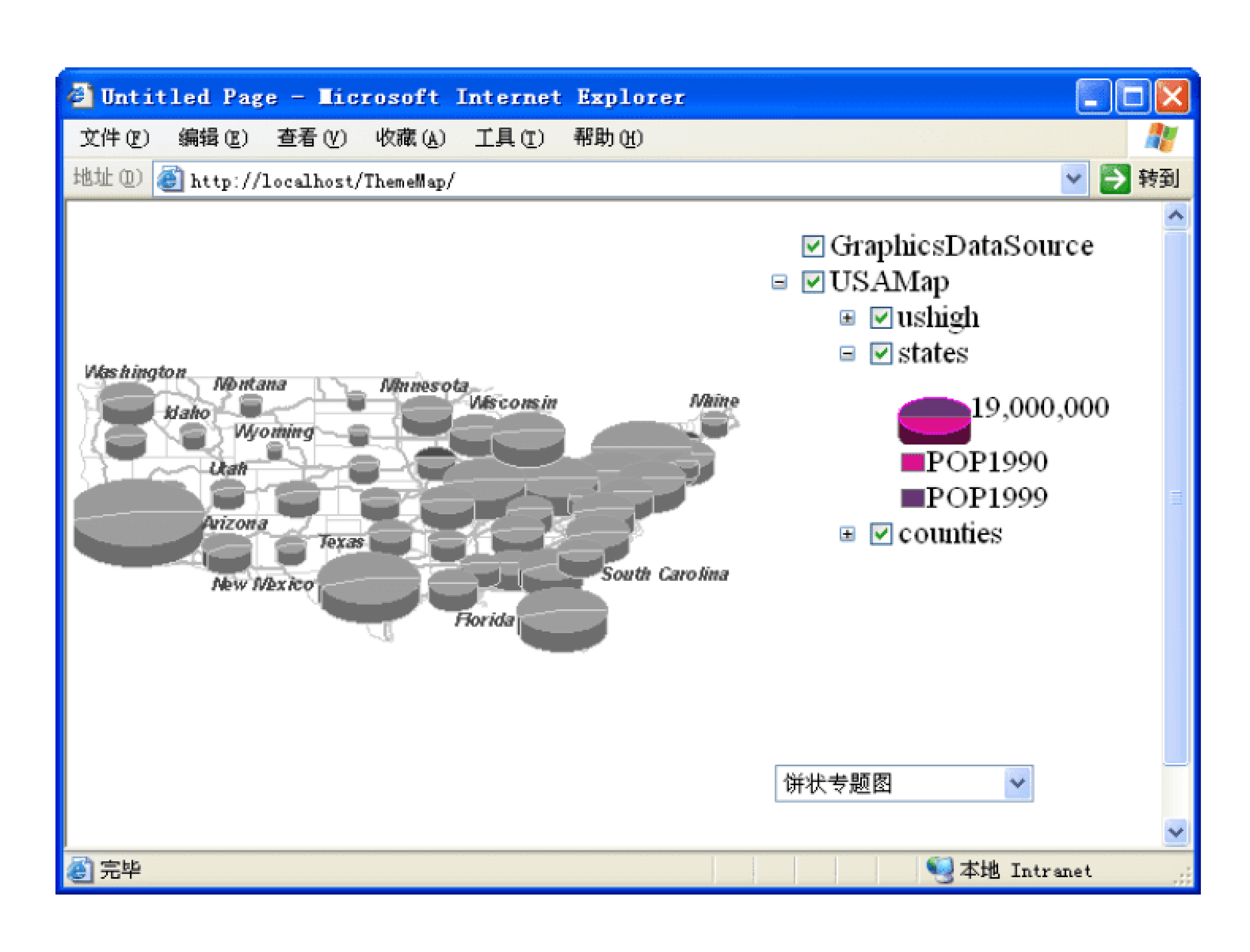


图 7.11　饼图专题图

7.4.5　等级符号专题图

等级符号专题图为表中每条记录显示一个符号，符号大小与数据值成比例。等级符号专题图用特定的数值来显示数据，对于阐明定量信息（如由高到低依次变化）很有用处。符号的大小与该要素每字段对应的数值成比例，数值越大符号就越大，数值越小符号就越小。因此，等级符号最适合数据值数据。例如使用等级符号图来显示不同区域的年销售额，或不同地区的人口数量等。

在 ArcGISObjects 中 IProportionalSymbolRenderer 接口可用于创建等级符号专题图。

我们利用名为 CreatePraportionalRenderer 的静态方法来实现等级符号专题图。首先在 GisFunctionality 类中增加上述方法的声明，代码如下：

```
    public static string CreateProportionalRenderer(
    ESRI.ArcGIS.ADF.Web.UI.WebControls.Map map,Toc toc, int layerID, string field)
    {
    }
```

从地图资源中得到服务器上下文，并从中得到图层对象。代码如下：

```
MapFunctionality gisfunc =
     map.GetFunctionality("USAMap") as MapFunctionality;
if (gisfunc == null)
    return "";

MapDescription mapDesc = gisfunc.MapDescription as MapDescription;
MapResourceLocal mapResLocal = gisfunc.MapResource as MapResourceLocal;
IServerContext serverContext = mapResLocal.ServerContextInfo.ServerContext;
if (serverContext == null)
    return "";

MapServer mapServer = mapResLocal.MapServer as MapServer;
IMapServerObjects mapServerObj = mapServer as IMapServerObjects;
string mapName = mapServer.get MapName(0);
ILayer layer = mapServerObj.get Layer(mapName, layerID);
IGeoFeatureLayer featureLyer = layer as IGeoFeatureLayer;
```

接着设置创建一填充符号，用于设置图层的背景，代码如下：

```
ISimpleFillSymbol                              symbol                              =
serverContext.CreateObject("esriDisplay.SimpleFillSymbol") as SimpleFillSymbol;
symbol.Color = GetRGB(serverContext, 239, 228, 249);
```

接着创建一简单的标志符号，用于设置图层中指定字段最小值的符号的大小与样式。代码如下：

```
ISimpleMarkerSymbol markSymbol = null;
markSymbol = serverContext.CreateObject("esriDisplay.SimpleMarkerSymbol")
    as ISimpleMarkerSymbol;
markSymbol.Color = GetRGB(serverContext, 190, 0, 170);
markSymbol.Style = esriSimpleMarkerStyle.esriSMSCircle;
markSymbol.Outline = false;
markSymbol.Size = 2.0;
```

需要对数据进行统计，得到最大值与最小值。统计代码如下：

```
ICursor pCursor = featureLyer.Search(null, true) as ICursor;
IDataStatistics pDataSta = new DataStatisticsClass();
pDataSta.Cursor = pCursor;
pDataSta.Field = field;
```

然后创建一等级符号着色器对象，设置其 Field、ValueUnit、MaxDataValue 以及 MinDataValue 等属性。代码如下：

```
IProportionalSymbolRenderer render = null;
render = serverContext.CreateObject("esriCarto.ProportionalSymbolRenderer")
    as IProportionalSymbolRenderer;
render.ValueUnit = esriUnits.esriUnknownUnits;
render.Field = field;
render.FlanneryCompensation = true;
render.LegendSymbolCount = 4;
render.MaxDataValue = pDataSta.Statistics.Maximum;
```

```
render.MinDataValue = pDataSta.Statistics.Minimum;
render.BackgroundSymbol = symbol;
render.MinSymbol = markSymbol as ISymbol;
render.CreateLegendSymbols();
```

接着将着色器应用到指定图层，并释放服务器上下文对象。代码如下：

```
featureLyer.Renderer = render as IFeatureRenderer;
serverContext.ReleaseContext();
```

最后刷新地图及 Toc。代码如下：

```
toc.Refresh();
CallbackResult tocCallbackResult = RefreshControlHtml(toc);
map.CallbackResults.Add(tocCallbackResult);
map.Refresh();
return map.CallbackResults.ToString();
```

切换到 Default.aspx.cs 文件中，找到 GetCallbackResult 方法，在 callbackArg 为 5 的部分，加入对 GisFunctionality 方法的调用。代码如下：

```
return GisFunctionality.CreateProportionalRenderer(Map1, Toc1, 1, "POP1999");
```

上述代码表示，针对 USAMap 地图资源中的第二个图层（即美国州行政区划）的各州 1999 年人口数据设置等级符号专题图。

编译并运行程序，程序运行效果如图 7.12 所示。

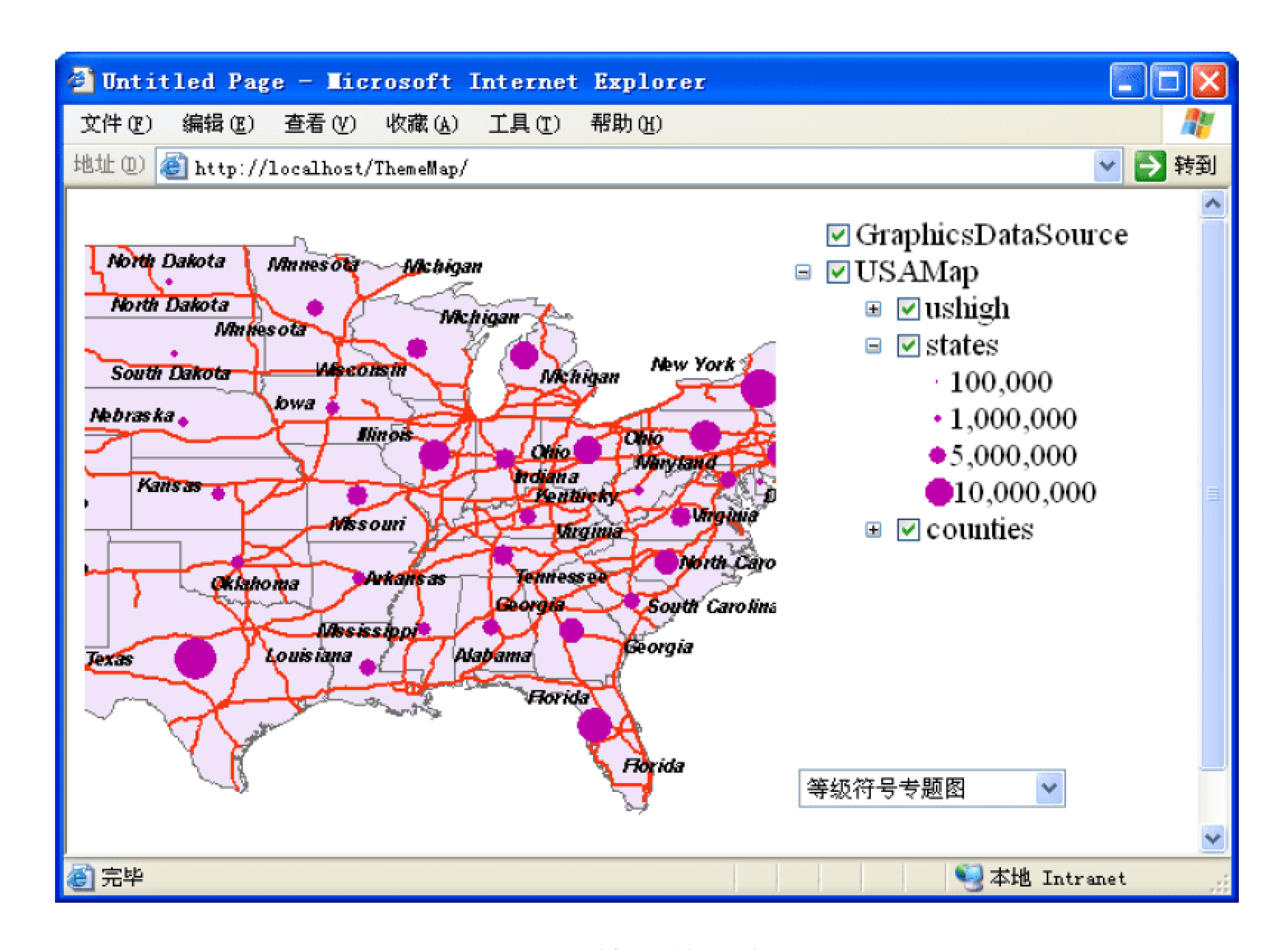


图 7.12　等级符号专题图

7.4.6　点密度专题图

点密度专题图则是在地图上用点来显示数据，每一点都代表一定数量，某区域中点的总数与该

区域数值成比例。每个点代表一定数量的单元，该数乘以区域内总的点数，就等于该区域的数据值。例如创建一个人口密度专题图，若某区域人口有 20000 人，而每个点代表 100 人，则该区域内将有 200 个点。

当要求在地图上显示一些原始数据，如人口数量、年销售额、犯罪率或者出生率等，点密度专题图十分有用。因此点密度专题图有着广泛的用途。

在 ArcGISObjects 中 IDotDensityRenderer 接口可用于创建等级符号专题图。

我们利用名为 CreateDotDensityRenderer 的静态方法来实现点密度专题图。首先在 GisFunctionality 类中增加上述方法的声明，代码如下：

```
    public static string CreateDotDensityRenderer (
    ESRI.ArcGIS.ADF.Web.UI.WebControls.Map map,Toc toc, int layerID, string field)
    {
    }
```

首先需要完成的还是从地图资源中得到服务器上下文，并从中得到图层对象。代码同前几个小节，这里不再重复。

接着设置创建一点密度填充符号，用于设置着色器的符号，代码如下：

```
    IDotDensityFillSymbol                          symbol                          =
serverContext.CreateObject("esriDisplay.DotDensityFillSymbol")                  as
IDotDensityFillSymbol;
    symbol.BackgroundColor = GetRGB(serverContext, 239, 228, 249);
    symbol.DotSize = 3;
```

上述只设置了点密度填充符号的大小，以及背景颜色。还需要另外创建一个简单标志符号来设置点的符号。代码如下：

```
    ISymbolArray symbolArray = symbol as ISymbolArray;
    ISimpleMarkerSymbol markSymbol = null;
    markSymbol = serverContext.CreateObject("esriDisplay.SimpleMarkerSymbol")
        as ISimpleMarkerSymbol;
    markSymbol.Color = GetRGB(serverContext, 190, 0, 170);
    markSymbol.Style = esriSimpleMarkerStyle.esriSMSCircle;
    markSymbol.Outline = false;
    markSymbol.Size = 1.0;
    symbolArray.AddSymbol((ISymbol)markSymbol);
```

接着创建点密度着色器对象，应用点密度填充符号，并设置每个点代表的数量。代码如下：

```
    IDotDensityRenderer render = null;
    render = serverContext.CreateObject("esriCarto.DotDensityRenderer")
        as IDotDensityRenderer;
    render.DotDensitySymbol = symbol;
    render.DotValue = 200000;
    render.ControlLayer = featureLyer;
    render.CreateLegend();
```

还需要加入字段的设置。代码如下：

```
    IRendererFields renderFields = render as IRendererFields;
```

```
    renderFields.AddField(field, field);
```

将该着色器应用到图层以及刷新地图与 Toc 控件的代码，与前几小节的相同。

切换到 Default.aspx.cs 文件中，找到 GetCallbackResult 方法，在 callbackArg 为 6 的部分，加入对 CreateDotDensityRenderer 方法的调用。代码如下：

```
    return GisFunctionality.CreateDotDensityRenderer(Map1, Toc1, 1, "POP1999");
```

上述代码表示，针对 USAMap 地图资源中的第二个图层（即美国州行政区划）的各州 1999 年人口数据设置点密度专题图。

编译并运行程序，程序运行效果如图 7.13 所示，从图中可以清楚地看出，美国东部人口最密集，其次是西部，再其次是南部与中部，北部人口最稀疏。



图 7.13　点密度专题图

7.5　图形对象的转换

由于 Web ADF 中在同一应用程序中，可以同时使用来自不同数据源的数据，而且每个数据源可以独立于 Web ADF，利用自己特有的 API 进行操作，因此应用程序既要处理来自客户端的图形对象，又要处理来自 GIS 服务器端数据源的图形对象，所以需要开发者负责转换这些不同类型的数据。在这一节中，将介绍 Web ADF 在数据类型转换方面提供的类与方法。

Web ADF 中提供了各种转换类，在不同的命名空间中以静态方法的方式提供。命名空间的名称表明了该转换类处理的数据类型。表 7-1 列出了每个转换类的功能。

表 7-1　不同命名空间中转换类的功能

类	描述
ESRI.ArcGIS.ADF.ArcGISServer.Converter	处理 ArcGIS Server 的 Web 服务与 DCOM 代理类的转换
ESRI.ArcGIS.ADF.Converter	处理 Web ADF 中的数据类型
ESRI.ArcGIS.ADF.Web.Converter	将.NET 数据集转换为 Web ADF 的数据集，将.NET 的数据表转换为 Web ADF 的图形图层
ESRI.ArcGIS.ADF.Web.DataSources.ArcGISServer.Converter	处理 ArcGIS Server 中 COM 与值对象类型
ESRI.ArcGIS.ADF.Web.DataSources.IMS.Converter	处理 ArcIMS 类型
ESRI.ArcGIS.ADF.Web.UI.WebControls.Converter	供 Web ADF 控件内部使用

7.5.1　几何类型的转换

GIS 应用和服务的功能都是和空间数据相关的，以处理和分析空间数据为主。比如获得鼠标点击的点，或者通过空间或属性查询选择要素等。在每个层次以及对于每个数据源，都必须管理空间信息，因此，每个层次与数据源都提供了存储与操作几何类型的方法。

假设我们要根据用户在浏览器中的地图上绘制的一个点，计算该点的缓冲区，并用该缓冲区来选择要素图层中的要素。该功能的实现可分四个步骤，如图 7.14 所示。

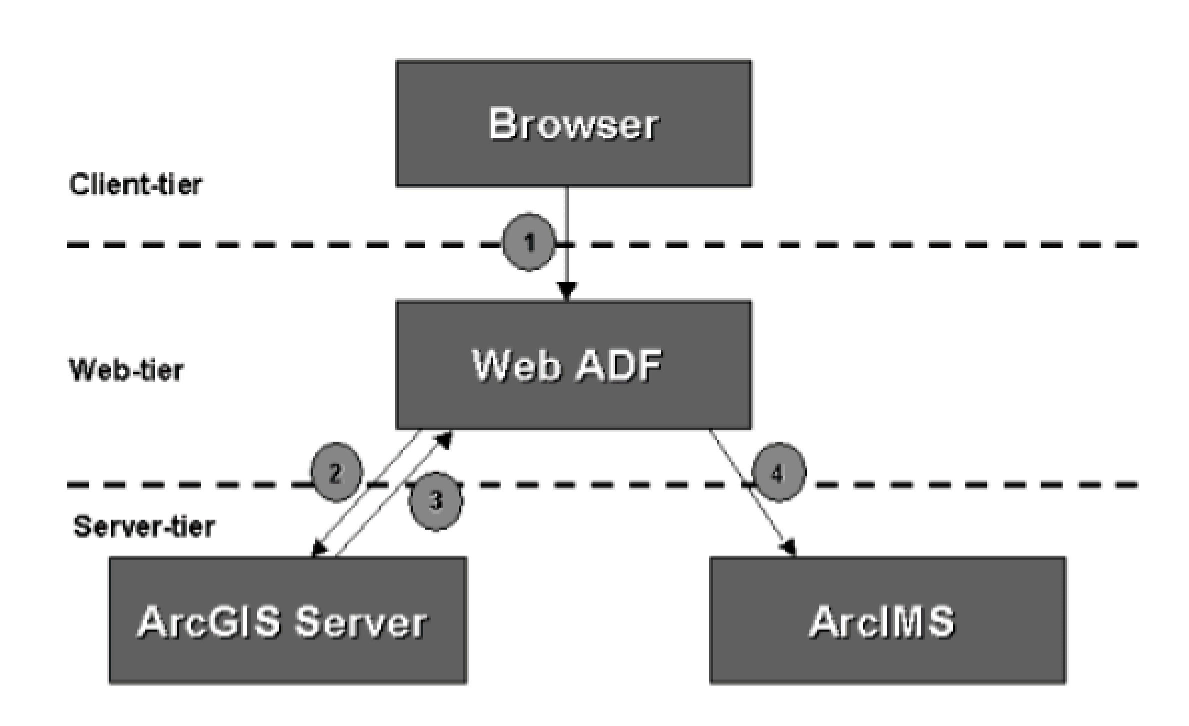


图 7.14　利用点缓冲区查询要素图层的步骤

（1）得到用户在浏览器的地图上单击的位置。用户在浏览器中输入的点使用的屏幕坐标，Web ADF 中使用.NET 的绘图类库（System.Drawing）存储屏幕几何类型。Web ADF 将该几何类型从屏幕坐标转换为地图坐标。Web ADF 使用自己的几何类型类库来操作 Web ADF 特有功能（例如 GraphicsElementLayer 或 SpatialFilter），扮演的是客户端到服务器端中间类型的脚色。

（2）使用 ArcGIS Server 本地服务计算点的缓冲区。Web ADF 不能计算点的缓冲区，但 ArcGIS Server 本地数据源可以计算几何类型的缓冲区，并返回一多边形几何类型。为了利用 ArcGIS Server 本地服务的缓冲区计算功能，我们需要将该 Web ADF 的点对象转换为 ArcObjects 的 COM 对象。

（3）在地图中的 Web ADF 图形图层中显示该缓冲区结果。ArcGIS Server 本地服务返回一个 ArcObjects 的 COM 类型的多边形，为了在 Web ADF 的图形图层中显示该多边形，需要将其转换

为 Web ADF 的多边形。

（4）使用缓冲区在 ArcIMS 服务中选择要素。为使用 Web ADF 的缓冲区多边形选择 ArcIMS 服务中的一个要素图层中的要素，需要将其转换为 ArcIMS 类型的多边形。如果仅仅需要查询一个图层，并返回一组要素，我们可以使用 Web ADF 中的 IQueryFunctionality 便可实现。然而我们希望针对某一特定的图层，因此需要使用 ArcIMS 的 API。

在上述步骤中，使用了 4 个不同的 API，在三个层次操作几何类型：.NET、Web ADF、ArcGIS Server（ArcObjects COM）与 ArcIMS。以下就来介绍不同 API 之间的几何类型的转换。

在下面的代码段中，使用了如下两个变量：

- tooleventargs，表示一自定义工具的 Web ADF 的 ToolEventArgs 参数;
- adf_map，表示 Web ADF 地图控件。

1. 点的转换

（1）将屏幕上的点转换为 Web ADF 中地图坐标，方法如下：

```
PointEventArgs pointargs = (PointEventArgs)tooleventargs;
System.Drawing.Point screen point = pointargs.ScreenPoint;
ESRI.ArcGIS.ADF.Web.Geometry.Point adf_point =
  ESRI.ArcGIS.ADF.Web.Geometry.Point.ToMapPoint(screen_point.X,
  screen point.Y,
  adf_map.GetTransformationParams(TransformationDirection.ToMap));
```

（2）将屏幕上的点转换为 ArcGIS Server SOAP 的点，方法如下：

```
ESRI.ArcGIS.ADF.ArcGISServer.ImageDisplay imgDisp =
      new ESRI.ArcGIS.ADF.ArcGISServer.ImageDisplay();
imgDisp.ImageHeight = (int)adf_map.Height.Value;
imgDisp.ImageWidth = (int)adf_map.Width.Value;
int[] xvalues = new int[1];
xvalues[0] = screen point.X;
int[] yvalues = new int[1];
yvalues[0] = screen point.Y;

ESRI.ArcGIS.ADF.Web.DataSources.ArcGISServer.MapFunctionality
ags_mapfunctionality = adf_map.GetFunctionality(0)
     as ESRI.ArcGIS.ADF.Web.DataSources.ArcGISServer.MapFunctionality;

ESRI.ArcGIS.ADF.Web.DataSources.ArcGISServer.MapResourceBase
ags mapresource = ags mapfunctionality.Resource
    as ESRI.ArcGIS.ADF.Web.DataSources.ArcGISServer.MapResourceBase;

ESRI.ArcGIS.ADF.ArcGISServer.MapServerProxy ags mapserverproxy =
     ags_mapresource.MapServerProxy;

ESRI.ArcGIS.ADF.ArcGISServer.MultipointN value multipoint =
  ags mapserverproxy.ToMapPoints(
          ags mapfunctionality.MapDescription,imgDisp, xvalues, yvalues)
          as ESRI.ArcGIS.ADF.ArcGISServer.MultipointN;
```

```
    ESRI.ArcGIS.ADF.ArcGISServer.PointN value_point =
      value_multipoint.PointArray[0] as ESRI.ArcGIS.ADF.ArcGISServer.PointN;
```

（3）屏幕上的点转换到 ArcIMS 的点，方法如下：

```
    ESRI.ArcGIS.ADF.Web.DataSources.IMS.MapFunctionality ims mapfunctionality =
      (ESRI.ArcGIS.ADF.Web.DataSources.IMS.MapFunctionality)
      adf map.GetFunctionality(0);

    ESRI.ArcGIS.ADF.IMS.Carto.MapView mapview = ims_mapfunctionality.MapView;
    ESRI.ArcGIS.ADF.IMS.Geometry.Envelope ims extent = mapview.Extent;
    ESRI.ArcGIS.ADF.IMS.Geometry.Point ims point =
      ESRI.ArcGIS.ADF.IMS.Geometry.Point.ToMapPoint(
                              screen point, ims extent,
                              mapview.ImageDescriptor.Width,
                              mapview.ImageDescriptor.Height);
```

（4）将 Web ADF 的点转换为 ArcObjects 的点，方法如下：

```
    ESRI.ArcGIS.ADF.Web.DataSources.ArcGISServer.MapFunctionality
ags mapfunctionality = adf map.GetFunctionality(ags local resource index)
     as ESRI.ArcGIS.ADF.Web.DataSources.ArcGISServer.MapFunctionality;
    ESRI.ArcGIS.ADF.Web.DataSources.ArcGISServer.MapResourceLocal
ags mapresourcelocal = ags mapfunctionality.Resource
     as ESRI.ArcGIS.ADF.Web.DataSources.ArcGISServer.MapResourceLocal;
    ESRI.ArcGIS.Geometry.IPoint com_point = (ESRI.ArcGIS.Geometry.IPoint)

ESRI.ArcGIS.ADF.Web.DataSources.ArcGISServer.Converter.ToIGeometry(adf point,
       ags_mapresourcelocal.ServerContextInfo.ServerContext);
```

（5）Web ADF 的点转换为 ArcIMS 的点，方法如下：

```
    ESRI.ArcGIS.ADF.IMS.Geometry.Point ims point =
     (ESRI.ArcGIS.ADF.IMS.Geometry.Point)ESRI.ArcGIS.ADF.Web.DataSources.IMS.C
onverter.ToIMSGeometry(adf_point);
```

（6）ArcGIS Server SOAP 的点转换为 Web ADF 的点，方法如下：

```
    ESRI.ArcGIS.ADF.Web.Geometry.Point new adf point =
      ESRI.ArcGIS.ADF.Web.DataSources.ArcGISServer.Converter.ToAdfPoint(value
point);
```

（7）ArcGIS Server ArcObjects 点转换为 Web ADF 点，方法如下：

```
    ESRI.ArcGIS.ADF.Web.Geometry.Point new adf point =
       ESRI.ArcGIS.ADF.Web.DataSources.ArcGISServer.Converter.FromIPoint(com_
point);
```

（8）ArcIMS 点转换为 Web ADF 点，方法如下：

```
    ESRI.ArcGIS.ADF.Web.Geometry.Point new adf point =
      (ESRI.ArcGIS.ADF.Web.Geometry.Point)
       ESRI.ArcGIS.ADF.Web.DataSources.IMS.Converter.ToADFGeometry(ims_point);
```

2. 线的转换

（1）屏幕转换到 Web ADF，方法如下：

```
VectorEventArgs vectorargs = (VectorEventArgs)tooleventargs;
System.Drawing.Point[] screen_points = vectorargs.Vectors;

ESRI.ArcGIS.ADF.Web.Geometry.PointCollection adf pointcollection =
   new ESRI.ArcGIS.ADF.Web.Geometry.PointCollection();

foreach (System.Drawing.Point screen point in screen points)
{
   adf_pointcollection.Add(ESRI.ArcGIS.ADF.Web.Geometry.Point.ToMapPoint
      (screen_point, adf_map.Extent,
      adf_map.GetTransformationParams(TransformationDirection.ToMap));
}

ESRI.ArcGIS.ADF.Web.Geometry.Path adf path =
   new ESRI.ArcGIS.ADF.Web.Geometry.Path();
adf path.Points = adf pointcollection;
ESRI.ArcGIS.ADF.Web.Geometry.PathCollection adf_paths =
   new ESRI.ArcGIS.ADF.Web.Geometry.PathCollection();
adf paths.Add(adf path);
ESRI.ArcGIS.ADF.Web.Geometry.Polyline adf polyline =
   new ESRI.ArcGIS.ADF.Web.Geometry.Polyline();
adf_polyline.Paths = adf_paths;
```

（2）屏幕转换到 ArcGIS Server SOAP，方法如下：

```
   ESRI.ArcGIS.ADF.ArcGISServer.ImageDisplay imgDisp =
      new ESRI.ArcGIS.ADF.ArcGISServer.ImageDisplay();
   imgDisp.ImageHeight = (int)adf_map.Height.Value;
   imgDisp.ImageWidth = (int)adf map.Width.Value;

   int[] xvalues = new int[screen points.Length];
   int[] yvalues = new int[screen points.Length];

   for (int i = 0; i < screen_points.Length; i++)
   {
      xvalues[i] = screen points[i].X;
      yvalues[i] = screen points[i].Y;
   }

   ESRI.ArcGIS.ADF.Web.DataSources.ArcGISServer.MapFunctionality
ags mapfunctionality = adf map.GetFunctionality(0)
   as ESRI.ArcGIS.ADF.Web.DataSources.ArcGISServer.MapFunctionality;

   ESRI.ArcGIS.ADF.Web.DataSources.ArcGISServer.MapResourceBase
ags mapresource =
           (ESRI.ArcGIS.ADF.Web.DataSources.ArcGISServer.MapResourceBase)ags
mapfunctionality.Resource;
```

```
    ESRI.ArcGIS.ADF.ArcGISServer.MapServerProxy ags_mapserverproxy =
        ags_mapresource.MapServerProxy;

    ESRI.ArcGIS.ADF.ArcGISServer.MultipointN value_multipoint =
        (ESRI.ArcGIS.ADF.ArcGISServer.MultipointN)
        ags_mapserverproxy.ToMapPoints(ags_mapfunctionality.MapDescription,
imgDisp, xvalues, yvalues);

    ESRI.ArcGIS.ADF.ArcGISServer.Path value_path =
        new ESRI.ArcGIS.ADF.ArcGISServer.Path();
    value_path.PointArray = value_multipoint.PointArray;

    ESRI.ArcGIS.ADF.ArcGISServer.Path[] value_paths =
        new ESRI.ArcGIS.ADF.ArcGISServer.Path[1];
    value_paths.SetValue(value_path, 0);

    ESRI.ArcGIS.ADF.ArcGISServer.PolylineN value_polyline =
        new ESRI.ArcGIS.ADF.ArcGISServer.PolylineN();
    value_polyline.PathArray = value_paths;
```

（3）屏幕转换到 ArcIMS，方法如下：

```
    ESRI.ArcGIS.ADF.Web.DataSources.IMS.MapFunctionality ims_mapfunctionality =
       (ESRI.ArcGIS.ADF.Web.DataSources.IMS.MapFunctionality)
       adf_map.GetFunctionality(0);
    ESRI.ArcGIS.ADF.IMS.Carto.MapView mapview = ims_mapfunctionality.MapView;
    ESRI.ArcGIS.ADF.IMS.Geometry.Envelope ims_extent = mapview.Extent;
    ESRI.ArcGIS.ADF.IMS.Geometry.PointCollection ims_pointcollection =
       new ESRI.ArcGIS.ADF.IMS.Geometry.PointCollection();

    foreach (System.Drawing.Point screen_point in screen_points)
    {
       ESRI.ArcGIS.ADF.IMS.Geometry.Point ims_point =
          ESRI.ArcGIS.ADF.IMS.Geometry.Point.ToMapPoint(screen_point,
                          ims_extent,
                          mapview.ImageDescriptor.Width,
                          mapview.ImageDescriptor.Height);
       ims_pointcollection.Add(ims_point);
    }

    ESRI.ArcGIS.ADF.IMS.Geometry.Path ims_path =
       new ESRI.ArcGIS.ADF.IMS.Geometry.Path();
    ims_path.Points = ims_pointcollection;

    ESRI.ArcGIS.ADF.IMS.Geometry.Polyline ims_polyline =
       new ESRI.ArcGIS.ADF.IMS.Geometry.Polyline();
    ims_polyline.Paths.Add(ims_path);
```

（4）Web ADF 转换到 ArcGIS Server SOAP，方法如下：

```
    ESRI.ArcGIS.ADF.ArcGISServer.PolylineN value_polyline =
```

```
       ESRI.ArcGIS.ADF.Web.DataSources.ArcGISServer.Converter.FromAdfPolyline(
adf_polyline);
```

（5）Web ADF 转换到 ArcGIS Server ArcObjects，方法如下：

```
    ESRI.ArcGIS.ADF.Web.DataSources.ArcGISServer.MapFunctionality
ags mapfunctionality                                                          =
(ESRI.ArcGIS.ADF.Web.DataSources.ArcGISServer.MapFunctionality)
       adf map.GetFunctionality(0);

    ESRI.ArcGIS.ADF.Web.DataSources.ArcGISServer.MapResourceLocal
ags mapresourcelocal                                                          =
(ESRI.ArcGIS.ADF.Web.DataSources.ArcGISServer.MapResourceLocal)
       ags mapfunctionality.Resource;

    ESRI.ArcGIS.Geometry.IPointCollection    com polyline pointcollection     =
(ESRI.ArcGIS.Geometry.IPointCollection)

ags_mapresourcelocal.ServerContextInfo.ServerContext.CreateObject("esriGeometr
y.Polyline");

    object Missing = Type.Missing;

    foreach (ESRI.ArcGIS.ADF.Web.Geometry.Path new adf path
    in adf_polyline.Paths)
    {
       ESRI.ArcGIS.Geometry.IPointCollection com_pointcollection =
            (ESRI.ArcGIS.Geometry.IPointCollection)ags mapresourcelocal.Server
ContextInfo.ServerContext.CreateObject("esriGeometry.Path");

       foreach (ESRI.ArcGIS.ADF.Web.Geometry.Point new adf point
       in new adf path.Points)
       {
          ESRI.ArcGIS.Geometry.IPoint com_point = (ESRI.ArcGIS.Geometry.IPoint)
ESRI.ArcGIS.ADF.Web.DataSources.ArcGISServer.Converter.ToIGeometry(new_adf_poi
nt,
             ags mapresourcelocal.ServerContextInfo.ServerContext);
          com pointcollection.AddPoint(com point, ref Missing, ref Missing);
       }
       com polyline pointcollection.AddPointCollection(com pointcollection);
    }

    ESRI.ArcGIS.Geometry.IPolyline com polyline =
       (ESRI.ArcGIS.Geometry.IPolyline)com_polyline_pointcollection;
```

（6）Web ADF 转换到 ArcIMS，方法如下：

```
    ESRI.ArcGIS.ADF.IMS.Geometry.Polyline ims polyline =
        (ESRI.ArcGIS.ADF.IMS.Geometry.Polyline)ESRI.ArcGIS.ADF.Web.DataSources
.IMS.Converter.ToIMSGeometry(adf_polyline);
```

（7）ArcGIS Server SOAP 转换到 Web ADF，方法如下：

```
    ESRI.ArcGIS.ADF.Web.Geometry.Point new adf polyline =
         ESRI.ArcGIS.ADF.Web.DataSources.ArcGISServer.Converter.ToAdfPolyline(
value_polyline);
```

（8）ArcGIS Server ArcObjects 转换到 Web ADF，方法如下：

```
    ESRI.ArcGIS.Geometry.IPointCollection         com pointcollection         =
(ESRI.ArcGIS.Geometry.IPointCollection)com polyline;

    ESRI.ArcGIS.ADF.Web.Geometry.Point[]            new adf points            =
ESRI.ArcGIS.ADF.Web.DataSources.ArcGISServer.Converter.FromIPointCollection(co
m pointcollection);

    ESRI.ArcGIS.ADF.Web.Geometry.PointCollection new_adf_pointcollection = new
ESRI.ArcGIS.ADF.Web.Geometry.PointCollection();

    for (int i = 0; i < new adf points.Length - 1; i++) {
       new adf pointcollection.Add(new adf points[i]);
    }
    ESRI.ArcGIS.ADF.Web.Geometry.Path         new adf path         =         new
ESRI.ArcGIS.ADF.Web.Geometry.Path();

    new adf path.Points = new adf pointcollection;

    ESRI.ArcGIS.ADF.Web.Geometry.PathCollection new adf pathcollection =
        new ESRI.ArcGIS.ADF.Web.Geometry.PathCollection();
    adf pathcollection.Add(new adf path);

    ESRI.ArcGIS.ADF.Web.Geometry.Polyline new_adf_polyline =
        new ESRI.ArcGIS.ADF.Web.Geometry.Polyline();
    new_adf_polyline.Paths = new_adf_pathcollection;
```

（9）ArcIMS 转换到 Web ADF，方法如下：

```
    ESRI.ArcGIS.ADF.Web.Geometry.Polyline new adf polyline =
       (ESRI.ArcGIS.ADF.Web.Geometry.Polyline)ESRI.ArcGIS.ADF.Web.DataSources.
IMS.Converter.ToADFGeometry(ims_polyline);
```

3．多边形的转换

（1）屏幕转换到 Web ADF，方法：

```
    VectorEventArgs vectorargs = (VectorEventArgs)tooleventargs;
    System.Drawing.Point[] screen_points = vectorargs.Vectors;

    ESRI.ArcGIS.ADF.Web.Geometry.PointCollection adf pointcollection = new
                    ESRI.ArcGIS.ADF.Web.Geometry.PointCollection();
    foreach (System.Drawing.Point screen point in screen points) {
      adf pointcollection.Add(ESRI.ArcGIS.ADF.Web.Geometry.Point.ToMapPoint(
         screen_point, adf_map.Extent,
```

```
        adf_map.GetTransformationParams(TransformationDirection.ToMap));
    }

    ESRI.ArcGIS.ADF.Web.Geometry.Ring adf ring =
        new ESRI.ArcGIS.ADF.Web.Geometry.Ring();
    adf ring.Points = adf pointcollection;
    ESRI.ArcGIS.ADF.Web.Geometry.RingCollection adf rings =
        new ESRI.ArcGIS.ADF.Web.Geometry.RingCollection();
    adf_rings.Add(adf_ring);
    ESRI.ArcGIS.ADF.Web.Geometry.Polygon adf_polygon =
        new ESRI.ArcGIS.ADF.Web.Geometry.Polygon();
    adf_polygon.Rings = adf_rings;
```

（2）屏幕转换到 ArcGIS Server SOAP，方法如下：

```
    ESRI.ArcGIS.ADF.ArcGISServer.ImageDisplay imgDisp =
             new ESRI.ArcGIS.ADF.ArcGISServer.ImageDisplay();
    imgDisp.ImageHeight = (int)adf_map.Height.Value;
    imgDisp.ImageWidth = (int)adf_map.Width.Value;

    int[] xvalues = new int[screen points.Length];
    int[] yvalues = new int[screen points.Length];

    for (int i = 0; i < screen points.Length; i++) {
       xvalues[i] = screen_points[i].X;
       yvalues[i] = screen_points[i].Y;
    }

    ESRI.ArcGIS.ADF.Web.DataSources.ArcGISServer.MapFunctionality
ags mapfunctionality =
    (ESRI.ArcGIS.ADF.Web.DataSources.ArcGISServer.MapFunctionality)
adf map.GetFunctionality(0);

    ESRI.ArcGIS.ADF.Web.DataSources.ArcGISServer.MapResourceBase
ags_mapresource =
    (ESRI.ArcGIS.ADF.Web.DataSources.ArcGISServer.MapResourceBase)
ags mapfunctionality.Resource;

    ESRI.ArcGIS.ADF.ArcGISServer.MapServerProxy ags mapserverproxy =
       ags mapresource.MapServerProxy;

    ESRI.ArcGIS.ADF.ArcGISServer.MultipointN value_multipoint =
       (ESRI.ArcGIS.ADF.ArcGISServer.MultipointN)ags mapserverproxy.ToMapPoints
(ags mapfunctionality.MapDescription,imgDisp, xvalues, yvalues);

    ESRI.ArcGIS.ADF.ArcGISServer.Ring         value ring          =          new
ESRI.ArcGIS.ADF.ArcGISServer.Ring();
    value ring.PointArray = value multipoint.PointArray;

    ESRI.ArcGIS.ADF.ArcGISServer.Ring[]        value_rings          =         new
ESRI.ArcGIS.ADF.ArcGISServer.Ring[1];
```

```
    value_rings.SetValue(value_ring, 0);

    ESRI.ArcGIS.ADF.ArcGISServer.PolygonN       value polygon       =       new
ESRI.ArcGIS.ADF.ArcGISServer.PolygonN();
    value_polygon.RingArray = value_rings;
```

（3）屏幕转换到 ArcIMS，方法如下：

```
    ESRI.ArcGIS.ADF.Web.DataSources.IMS.MapFunctionality   ims mapfunctionality
=(ESRI.ArcGIS.ADF.Web.DataSources.IMS.MapFunctionality)
adf_map.GetFunctionality(0);
    ESRI.ArcGIS.ADF.IMS.Carto.MapView mapview = ims mapfunctionality.MapView;

    ESRI.ArcGIS.ADF.IMS.Geometry.Envelope ims extent = mapview.Extent;

    ESRI.ArcGIS.ADF.IMS.Geometry.PointCollection   ims pointcollection   =   new
ESRI.ArcGIS.ADF.IMS.Geometry.PointCollection();

    foreach (System.Drawing.Point screen_point in screen_points)
    {
        ESRI.ArcGIS.ADF.IMS.Geometry.Point ims point =
          ESRI.ArcGIS.ADF.IMS.Geometry.Point.ToMapPoint(screen point,
                                 ims extent,
                                 mapview.ImageDescriptor.Width,
                                 mapview.ImageDescriptor.Height);
        ims_pointcollection.Add(ims_point);
    }

    ESRI.ArcGIS.ADF.IMS.Geometry.Ring ims ring =
         new ESRI.ArcGIS.ADF.IMS.Geometry.Ring();
    ims ring.Points = ims pointcollection;

    ESRI.ArcGIS.ADF.IMS.Geometry.Polygon ims_polygon =
         new ESRI.ArcGIS.ADF.IMS.Geometry.Polygon();
    ims_polygon.Rings.Add(ims_ring);
```

（4）Web ADF 转换到 ArcGIS Server SOAP 方法如下：

```
    ESRI.ArcGIS.ADF.ArcGISServer.PolygonN           value polygon            =
ESRI.ArcGIS.ADF.Web.DataSources.ArcGISServer.Converter.FromAdfPolygon(adf poly
gon);
```

（5）Web ADF 转换到 ArcGIS Server ArcObjects，方法如下：

```
    ESRI.ArcGIS.ADF.Web.DataSources.ArcGISServer.MapFunctionality
ags mapfunctionality =
    (ESRI.ArcGIS.ADF.Web.DataSources.ArcGISServer.MapFunctionality)
adf map.GetFunctionality(0);

    ESRI.ArcGIS.ADF.Web.DataSources.ArcGISServer.MapResourceLocal
ags mapresourcelocal =
    (ESRI.ArcGIS.ADF.Web.DataSources.ArcGISServer.MapResourceLocal)
```

Web GIS

```
ags_mapfunctionality.Resource;

    ESRI.ArcGIS.Geometry.IPolygon com_polygon = (ESRI.ArcGIS.Geometry.IPolygon)
ESRI.ArcGIS.ADF.Web.DataSources.ArcGISServer.Converter.ToIGeometry(adf_polygon,
ags_mapresourcelocal.ServerContextInfo.ServerContext);
```

（6）Web ADF 转换到 ArcIMS，方法如下：

```
    ESRI.ArcGIS.ADF.IMS.Geometry.Polygon ims_polygon =
         (ESRI.ArcGIS.ADF.IMS.Geometry.Polygon)ESRI.ArcGIS.ADF.Web.DataSources
.IMS.Converter.ToIMSGeometry(adf_polygon);
```

（7）ArcGIS Server SOAP 转换到 Web ADF，方法如下：

```
    ESRI.ArcGIS.ADF.Web.Geometry.Polygon new_adf_polygon =
          ESRI.ArcGIS.ADF.Web.DataSources.ArcGISServer.Converter.ToAdfPolygon(
value_polygon);
```

（8）ArcGIS Server ArcObjects 转换到 Web ADF，方法如下：

```
    ESRI.ArcGIS.Geometry.IPointCollection com_pointcollection =
            (ESRI.ArcGIS.Geometry.IPointCollection)com_polygon;
    ESRI.ArcGIS.ADF.Web.Geometry.Point[] new_adf_points =
             ESRI.ArcGIS.ADF.Web.DataSources.ArcGISServer.Converter.FromIPoint
Collection(com_pointcollection);
    ESRI.ArcGIS.ADF.Web.Geometry.PointCollection new_adf_pointcollection = new
ESRI.ArcGIS.ADF.Web.Geometry.PointCollection();

    for (int i = 0; i < new_adf_points.Length - 1; i++)
    {
       new_adf_pointcollection.Add(new_adf_points[i]);
    }
    ESRI.ArcGIS.ADF.Web.Geometry.Ring new_adf_ring =
       new ESRI.ArcGIS.ADF.Web.Geometry.Ring();

    new_adf_ring.Points = new_adf_pointcollection;
    ESRI.ArcGIS.ADF.Web.Geometry.RingCollection new_adf_ringcollection =
       new ESRI.ArcGIS.ADF.Web.Geometry.RingCollection();
    new_adf_ringcollection.Add(new_adf_ring);
    ESRI.ArcGIS.ADF.Web.Geometry.Polygon new_adf_polygon =
       new ESRI.ArcGIS.ADF.Web.Geometry.Polygon();
    new_adf_polygon.Rings = new_adf_ringcollection;
```

（9）ArcIMS 转换到 Web ADF 方法如下：

```
    ESRI.ArcGIS.ADF.Web.Geometry.Polygon new_adf_polygon =
         (ESRI.ArcGIS.ADF.Web.Geometry.Polygon)ESRI.ArcGIS.ADF.Web.DataSources
.IMS.Converter.ToADFGeometry(ims_polygon);
```

7.5.2 ArcGIS Server 中 COM 对象与值对象的转换

当使用 ArcGIS Server 时，Web ADF 控件与公有 API 使用无状态的 ArcGIS Server SOAP API，该 API 中包含了一组值对象与代理类。ArcGIS Server 远程数据源使用 Web 服务代理，将值对象序列化为 SOAP，并通过 IRequestHandler 接口，直接使用服务器对象。不管是通过远程连接还是本地连接访问服务器对象的 SOAP 接口，ArcGIS Server SOAP API 的使用与限制是一样的。然而，如果使用 ArcGIS Server 本地数据源，便可以访问服务器上下文，也就可以通过 GIS 服务器上的 COM，访问 ArcObjects API。

ArcGIS Server SOAP API 只提供了 ArcGIS Server ArcObjects API 部分功能。由于 Web ADF 使用 SOAP API 实现 ArcGIS Server 数据源，因此为了利用 ArcObjects，就需要在 ArcObjects 的 COM 对象与 SOAP 的值对象之间进行转换。

ESRI.ArcGIS.ADF.Web.DataSources.ArcGISServer.Converter 类中提供了两个方法来实现上述转换：

（1）COMObjectToValueObject(object, ESRI.ArcGIS.Server.IServerContext, System.Type)——返回一个 ArcGIS Server SOAP API 值对象；

```
    ESRI.ArcGIS.ADF.ArcGISServer.PolylineN value polyline =
          (ESRI.ArcGIS.ADF.ArcGISServer.PolylineN)ESRI.ArcGIS.ADF.ArcGISServer.
Converter.ComObjectToValueObject(com_polyline,servercontext,typeof(ESRI.ArcGIS
.ADF.ArcGISServer.PolylineN));
```

（2）ValueObjectToCOMObject(object, ESRI.ArcGIS.Server.IServerContext)——返回一 ArcGIS Server ArcObjects API COM 对象。

```
    ESRI.ArcGIS.Geometry.IPolyline com polyline =
         (ESRI.ArcGIS.Geometry.IPolyline)ESRI.ArcGIS.ADF.ArcGISServer.Converter
.ValueObjectToComObject(value_polyline, servercontext);
```

由于 SOAP API 是 ArcObjects API 的一部分，因此每个值对象有一个对应的 COM 对象。但是，不是所有的 COM 对象都有一个对应的值对象。表 7-2 列出了值对象与 COM 对象之间的有效转换。

表 7-2　值对象与 COM 对象之间的有效转换

值对象类型	ArcObjects 的 COM 对象类型
PointN	ESRI.ArcGIS.Geometry.IPoint
Line	ESRI.ArcGIS.Geometry.ILine
PolylineN	ESRI.ArcGIS.Geometry.IPolyline
PolygonN	ESRI.ArcGIS.Geometry.IPolygon
MapDescription	ESRI.ArcGIS.Carto.IMapDescription
GraphicsElement[]	ESRI.ArcGIS.Carto.IGraphicElements
Field	ESRI.ArcGIS.Geodatabase.IField
RecordSet	ESRI.ArcGIS.Geodatabase.IRecordSet

7.5.3 数据集转换

操作数据源时，常常需要处理表格形式的数据。在大多数情况下，数据表的转换需要使用数据源特有的 API 方法、属性与技术。为有效处理数据源特有的数据集，以及操作通用的标准的 ADO.NET 数据结构，Web ADF 提供了一组功能方法。这些方法可用于将数据源特有的表格数据转换为 ADO.NET 的 DataTable 与 Web ADF 的 GraphicsLayer 类型。

为将 ArcGIS Server ArcObjects 的 ESRI.ArcGIS.Geodatabase.IRecordSet 或 ESRI.ArcGIS.ADF.ArcGISServer.RecordSet 转换为 ADO.NET 的 System.Data.DataTable，可使用 ESRI.ArcGIS.ADF.Web.DataSources.ArcGISServer 命名空间中的如下静态方法。在转换之前，需要使用前一中介绍的 ComObjectToValueObject 方法，将 ArcObjects 类型的转换为一 SOAP 值对象，然后使用如下方法：

```
ToDataTable(ESRI.ArcGIS.ADF.ArcGISServer.RecordSet)
```

如果 ArcGIS Server RecordSet（记录集）值对象中的一列包含几何类型值对象，可将该 RecordSet 转换为 Web ADF 要素图形图层，在地图控件中显示。使用下面的静态方法返回一个 ESRI.ArcGIS.ADF.Web.Display.Graphics.FeatureGraphicsLayer 对象：

```
ToFeatureGraphicsLayer(ESRI.ArcGIS.ADF.ArcGISServer.RecordSet recordSet)
```

Web ADF 公有 API 定义了一个 IQueryFunctionality 接口，如果某数据源实现了该接口，那么就可用来查询该数据源的数据图层。查询返回的数据的格式是 ADO.NET DataTable 对象。然而，该对象通常包含 Web ADF 几何类型，可用来在地图控件中显示。这时不需要从头创建 Web ADF 图形图层，可使用 ESRI.ArcGIS.ADF.Web.Converter 类的如下静态方法创建图形图层：

```
ToGraphicsLayer(System.Data.DataTable)
```

如果在数据表中的所有 Web ADF 几何类型同一类型，那么返回的是一 ESRI.ArcGIS.ADF.Web.Display.Graphics.FeatureGraphicsLayer。如果属于不同类型，返回的是一 ESRI.ArcGIS.ADF.Web.Display.Graphics.ElementGraphicsLayer。不管是哪种情况，都可以将该图形图层加入到 Web ADF 的图形资源中。下面的代码演示了如何完成上述功能：

```
System.Data.DataTable datatable = queryfunctionality.Query(
                          null, layerids[layer_index], adf_spatialfilter);
ESRI.ArcGIS.ADF.Web.Display.Graphics.GraphicsLayer graphicslayer =
   Converter.ToGraphicsLayer(datatable, Color.Yellow, Color.Yellow);
ESRI.ArcGIS.ADF.Web.DataSources.Graphics.MapResource graphicsresource =
   null;
foreach (IGISFunctionality gisfunctionality in gisfunctionalitycollection)
{
   if (gisfunctionality.Resource.Name == "Selection") {
      graphicsresource =
              (ESRI.ArcGIS.ADF.Web.DataSources.Graphics.MapResource)gfunc
.Resource;
      graphicsresource.Graphics.Tables.Clear();
   }
```

```
    }
    graphicsresource.Graphics.Tables.Add(graphicslayer);
```

7.5.4 缓冲区分析

所谓缓冲区就是地理空间目标的一种影响范围或服务范围。缓冲区分析是 GIS 中最重要的空间分析之一。本节将通过点、线、多边形的缓冲区分析，介绍图形对象在客户端、Web 端以及 GIS 服务器端之间的转换。

在 Visual Studio 2005 中，利用 File 菜单的 New Web Site 创建一个新的站点，命名为 BufferAnalysis。

在 Default.aspx 页面中增加一地图资源管理器控件、一地图控件、一工具栏控件与一 Toc 控件。通过地图资源管理器控件的 MapResourcesItem 属性设置对话框，加入 USAMap 地图资源。并通过工程的右键菜单的 Add ArcGIS Identity 命令，加入连接 USAMap 地图资源的身份信息。

按照 4.4 节中介绍的自定义工具的操作，在工具栏控件加入放大、缩小、漫游与全图工具或命令，并加入三个自定义工具，分别用于点、线与多边形的缓冲区分析。工具栏控件中这三个工具在 Default.aspx 中的代码如下：

```
<esri:Tool ClientAction="Point" Name="PointBuffer"
   DefaultImage="~/Images/point.gif" HoverImage="~/Images/pointU.gif"
   JavaScriptFile="" SelectedImage="~/Images/pointD.gif"
   ServerActionAssembly="App Code"
   ServerActionClass="PointBuffer"
   ToolTip="点缓冲区分析" />
<esri:Tool ClientAction="Polyline" Name="LineBuffer"
   DefaultImage="~/Images/polyline.gif"
   HoverImage="~/Images/polylineU.gif"
   SelectedImage="~/Images/polylineD.gif"
   ServerActionAssembly="App Code"
   ServerActionClass="LineBuffer"
   ToolTip="线缓冲区分析" JavaScriptFile="" />
<esri:Tool ClientAction="Polygon" Name="PolygonBuffer"
   DefaultImage="~/Images/polygon.gif"
   HoverImage="~/Images/polygonU.gif"
   SelectedImage="~/Images/polygonD.gif"
   ServerActionAssembly="App_Code"
   ServerActionClass="PolygonBuffer"
   ToolTip="多边形缓冲区分析" JavaScriptFile="" />
```

上述代码指明工具使用的图标位于工程的 Images 文件夹中，因此需要读者从本书配套的源代码文件中拷贝这些图片文件。代码也指明了每个工具处理类。

在工程中加入 ASP.NET 的标准文件夹 App_Code。

1．点缓冲区分析

在 App_Code 文件夹中新增加一个 C#类，命名为 PointBuffer。

在类的头部加入如下命名空间的引用：

```
using ESRI.ArcGIS.ADF.Web.Geometry;
using ESRI.ArcGIS.ADF.Web.UI.WebControls;
using ESRI.ArcGIS.ADF.Web.UI.WebControls.Tools;
using ESRI.ArcGIS.ADF.Web.DataSources.ArcGISServer;
```

修改类的声明代码，增加 IMapServerToolAction 接口的支持。代码如下：

```
public class PointBuffer : IMapServerToolAction
```

利用集成开发环境的代码自动生成功能，创建 IMapServerToolAction 接口的实现框架。该接口中只有一个方法，即：

```
void IMapServerToolAction.ServerAction(ToolEventArgs args) {
}
```

首先需要将屏幕坐标的点转换为地图坐标的点，代码如下：

```
// 从方法的参数中得到地图控件的引用
Map map = args.Control as Map;
// 从方法的参数中得到用户单击位置的点
PointEventArgs pea = (PointEventArgs)args;
System.Drawing.Point screen point = pea.ScreenPoint;
// 将屏幕坐标的点转换为地图坐标的点
Point adf_map_point = Point.ToMapPoint(screen_point.X, screen_point.Y,
    map.Extent, (int)map.Width.Value, (int)map.Height.Value);
```

然后利用如下代码将 Web ADF 的点对象转换为 ArcGIS Server 的点对象：

```
ESRI.ArcGIS.ADF.ArcGISServer.PointN ags map point;
ags map point =
       ESRI.ArcGIS.ADF.Web.DataSources.ArcGISServer.Converter.FromAdfPoint(
adf_map_point);
```

由于缓冲区分析需要使用的是 COM 对象，因此需要先得到服务器上下文对象。代码如下：

```
MapFunctionality mf;
mf = (MapFunctionality)map.GetFunctionality("USAMap");
if (mf == null)
    return;
MapResourceLocal mrl;
mrl = (MapResourceLocal)mf.MapResource;
ESRI.ArcGIS.Server.IServerContext serverContext =
    mrl.ServerContextInfo.ServerContext;
```

得到服务器上下文对象后，便可以将点对象转换为 COM 对象，然后利用 ITopologicalOperator 接口的 Buffer 方法，进行缓冲区分析，得到一多边形结果。由于该方法既可以用于点的缓冲区分析，又可以用于线与多边形的缓冲区分析，因此我们将该过程提取出来，单独形成一个静态方法。同时由于在 ArcGIS Server 中多个命名空间都有点、线、多边形等类，因此为了避免它们之间的冲突，我们将该方法放在另一个辅助类中。

再在 App_Code 文件夹中新增加一个 C#类，命名为 BufferHelper。

加入如下命名空间的引用：

```
using ESRI.ArcGIS.Carto;
using ESRI.ArcGIS.ADF.ArcGISServer;
using ESRI.ArcGIS.Geometry;
```

然后加入根据一个 Web ADF 的几何类型对象，进行缓冲区分析的方法。代码如下：

```
public static ESRI.ArcGIS.Geometry.IPolygon Buffer(
ESRI.ArcGIS.Server.IServerContext serverContext,
ESRI.ArcGIS.ADF.ArcGISServer.Geometry ags_map_geo)
{
    IGeometry geo =
        ESRI.ArcGIS.ADF.Web.DataSources.ArcGISServer.Converter.ValueObject
ToComObject(ags_map_geo, serverContext) as ESRI.ArcGIS.Geometry.IGeometry;

    ITopologicalOperator topop = (ITopologicalOperator)geo;
    double bufferdistance = 1;
    return (ESRI.ArcGIS.Geometry.IPolygon)topop.Buffer(bufferdistance);
}
```

上述代码中首先将 Web ADF 的几何类型对象，转换为 COM 对象，然后利用 ITopologicalOperator 接口的 Buffer 方法，进行缓冲区分析。

回到 PointBuffer 类的 ServerAction 方法中，在代码的最后加入对上述方法的调用，得到缓冲区分析结果。代码如下：

```
ESRI.ArcGIS.Geometry.IPolygon bufferpolygon =
        BufferHelper.Buffer(serverContext, ags_map_point);
```

然后需要实现的是根据该缓冲区多边形，对美国各县进行空间查询，得到与该多边形相交的要素。代码如下：

```
ESRI.ArcGIS.ADF.ArcGISServer.PolygonN[] queryResults =
        BufferHelper.Query(serverContext, bufferpolygon);
```

上行代码实际调用的是 BufferHelper 类的 Query 方法实现查询。该方法的实现代码如下：

```
public static PolygonN[] Query(
ESRI.ArcGIS.Server.IServerContext serverContext,
ESRI.ArcGIS.Geometry.IPolygon polygon)
{
    IMapServer mapServer = serverContext.ServerObject as IMapServer;

    IMapServerInfo mapInfo =
        mapServer.GetServerInfo(mapServer.DefaultMapName);
    IMapDescription mapDesc = mapInfo.DefaultMapDescription;

    IImageDisplay imgDisp = new ESRI.ArcGIS.Carto.ImageDisplay();
    imgDisp.Height = 500;
    imgDisp.Width = 500;
    imgDisp.DeviceResolution = 96;
```

```
        // 只对 ID 为 2 的图层进行查询，也就是美国县级行政区划图层
        ESRI.ArcGIS.esriSystem.ILongArray layerIds =
            new ESRI.ArcGIS.esriSystem.LongArray();
        layerIds.Add(2);

        IMapServerIdentifyResults results = mapServer.Identify(mapDesc, imgDisp,
polygon, 0, ESRI.ArcGIS.Carto.esriIdentifyOption.esriIdentifyAllLayers,
layerIds);

        PolygonN[] resultPolygons = new PolygonN[results.Count];
        for (int i = 0; i < results.Count; i++) {
            IMapServerIdentifyResult result = results.get Element(i);
            ESRI.ArcGIS.Geometry.IGeometry geo = result.Shape;
            resultPolygons[i] =
                ESRI.ArcGIS.ADF.Web.DataSources.ArcGISServer.Converter.ComObject
ToValueObject(geo, serverContext, typeof(PolygonN)) as PolygonN;
        }

        return resultPolygons;
    }
```

下面要完成的工作就是显示这些几何图形对象。我们这里使用服务器端的方法进行绘制。先得到 MapDescription 对象，代码如下：

```
    ESRI.ArcGIS.ADF.ArcGISServer.MapDescription mapDescription =
          mf.MapDescription;
```

我们要绘制的是用户单击的点、该点的缓冲区以及缓冲区查询结果，因此需要利用代码的代码创建图形元素对象：

```
    ESRI.ArcGIS.ADF.ArcGISServer.GraphicElement[] ges;
    ges =
        new ESRI.ArcGIS.ADF.ArcGISServer.GraphicElement[2 + queryResults.Length];
```

然后需要逐一设置这些图形元素对象。先设置缓冲区查询结果部分的图形要素对象。代码如下：

```
    ESRI.ArcGIS.ADF.ArcGISServer.SimpleFillSymbol querySym =
        AdfHelper.CreateSolidFillSymbol(255, 0, 0);
    for (int i = 0; i < queryResults.Length; i++)
    {
        ESRI.ArcGIS.ADF.ArcGISServer.PolygonElement queryPolygon;
        queryPolygon = new ESRI.ArcGIS.ADF.ArcGISServer.PolygonElement();
        queryPolygon.Symbol = querySym;
        queryPolygon.Polygon = queryResults[i];
        ges[i] = queryPolygon;
    }
```

在上述代码中，先利用 AdfHelper 类的 CreateSolidFillSymbol 方法创建一实填充符号，然后利用一个循环，绘制查询结果中的每个多边形。

在 App_Code 文件夹中新增加一个 C#类，命名为 AdfHelper。

在该类的头部加入如下命名空间的引用：

```
using ESRI.ArcGIS.ADF.Web.UI.WebControls;
using ESRI.ArcGIS.ADF.ArcGISServer;
```

然后加入创建实填充符号 CreateSolidFillSymbol 方法。代码如下：

```
public static SimpleFillSymbol CreateSolidFillSymbol(
byte red, byte green, byte blue) {
    SimpleFillSymbol querySym = new SimpleFillSymbol();
    querySym.Style = esriSimpleFillStyle.esriSFSSolid;
    querySym.Color = CreateColor(red, green, blue);
    return querySym;
}

public static RgbColor CreateColor(byte red, byte green, byte blue) {
    RgbColor rgb = new RgbColor();
    rgb.Red = red;
    rgb.Green = green;
    rgb.Blue = blue;
    rgb.AlphaValue = 255;
    return rgb;
}
```

回到 PointBuffer 类的 ServerAction 方法中，在代码的最后加入如下代码，将 COM 类型的缓冲区对象转换为 Web ADF 的多边形对象：

```
ESRI.ArcGIS.ADF.ArcGISServer.PolygonN buffer polyn;
buffer_polyn =
     (ESRI.ArcGIS.ADF.ArcGISServer.PolygonN)ESRI.ArcGIS.ADF.Web.DataSources
.ArcGISServer.Converter.ComObjectToValueObject(bufferpolygon, serverContext,
typeof(ESRI.ArcGIS.ADF.ArcGISServer.PolygonN));
```

然后加入绘制缓冲区多边形对象的代码。代码如下：

```
ESRI.ArcGIS.ADF.ArcGISServer.LineSymbol lnSymbol = null;
lnSymbol = new ESRI.ArcGIS.ADF.ArcGISServer.SimpleLineSymbol();
lnSymbol.Color = AdfHelper.CreateColor(0, 255, 0);
lnSymbol.Width = 1;
ESRI.ArcGIS.ADF.ArcGISServer.SimpleFillSymbol sfs1 =
             AdfHelper.CreateFillSymbol(255, 255, 0, lnSymbol);

ESRI.ArcGIS.ADF.ArcGISServer.PolygonElement bufferPolyElement;
bufferPolyElement = new ESRI.ArcGIS.ADF.ArcGISServer.PolygonElement();
bufferPolyElement.Symbol = sfs1;
bufferPolyElement.Polygon = buffer_polyn;
```

在上述代码中利用 AdfHelper 类的 CreateFillSymbol 静态方法，创建一交叉线填充的符号，然后创建一多边形元素。

AdfHelper 类的 CreateFillSymbol 静态方法的代码如下：

```
public static SimpleFillSymbol CreateFillSymbol(
byte red, byte green, byte blue, LineSymbol outline)
{
```

```
        SimpleFillSymbol querySym = new SimpleFillSymbol();
        querySym.Style = esriSimpleFillStyle.esriSFSCross;
        querySym.Color = CreateColor(red, green, blue);
        querySym.Outline = outline;
        return querySym;
    }
```

回到 PointBuffer 类的 ServerAction 方法中，在代码的最后加入绘制用户在地图上单击的点。代码如下：

```
    ESRI.ArcGIS.ADF.ArcGISServer.SimpleMarkerSymbol sms;
    sms = new ESRI.ArcGIS.ADF.ArcGISServer.SimpleMarkerSymbol();
    sms.Style =
        ESRI.ArcGIS.ADF.ArcGISServer.esriSimpleMarkerStyle.esriSMSCircle;
    sms.Color = AdfHelper.CreateColor(0, 255, 0);
    sms.Size = 5.0;

    ESRI.ArcGIS.ADF.ArcGISServer.MarkerElement marker;
    marker = new ESRI.ArcGIS.ADF.ArcGISServer.MarkerElement();
    marker.Symbol = sms;
    marker.Point = ags_map_point;
```

然后将缓冲区多边形元素与上述点元素设置给图形元素数组，代码如下：

```
    ges[queryResults.Length] = bufferPolyElement;
    ges[queryResults.Length + 1] = marker;
```

最后将该图形元素数组赋给 MapDescription 对象的 CustomGraphics 属性，并刷新地图。代码如下：

```
    mapDescription.CustomGraphics = ges;
    map.Refresh();
```

编译并运行程序，通过点缓冲区工具，在地图上单击，便可在地图上显示该点的缓冲区以及该缓冲区影响的县。效果如图 7.15 所示。

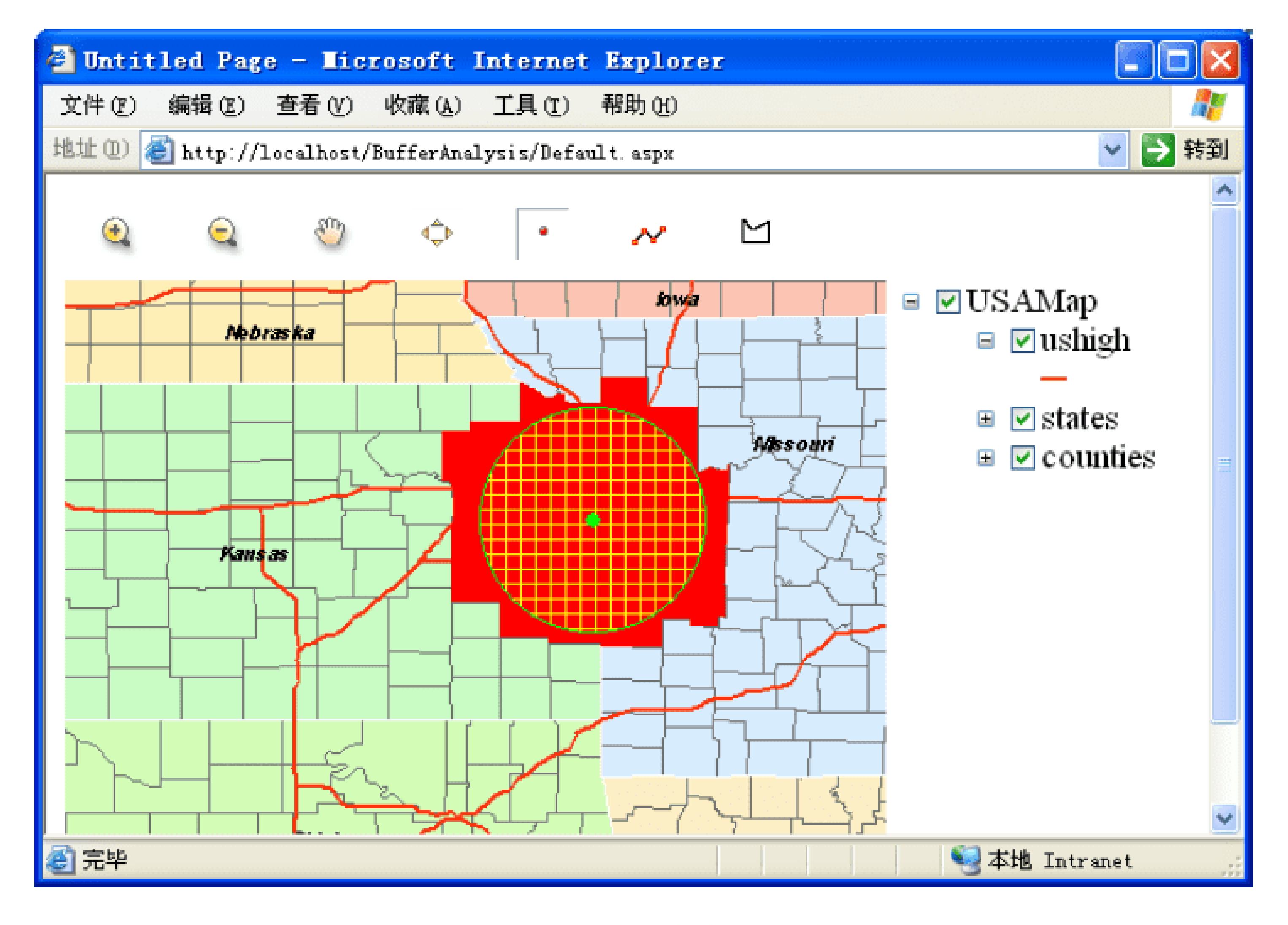


图 7.15 点缓冲区分析

2. 线缓冲区分析

在 App_Code 文件夹中新增加一 C#类，命名为 LineBuffer。

在类的头部加入如下命名空间的引用：

```
using ESRI.ArcGIS.ADF.Web.Geometry;
using ESRI.ArcGIS.ADF.Web.UI.WebControls;
using ESRI.ArcGIS.ADF.Web.UI.WebControls.Tools;
using ESRI.ArcGIS.ADF.Web.DataSources.ArcGISServer;
```

修改类的声明代码，增加对 IMapServerToolAction 接口的支持。代码如下：

```
public class LineBuffer : IMapServerToolAction
```

利用集成开发环境的代码自动生成功能，创建 IMapServerToolAction 接口的实现框架。该接口中同样只有一个方法，即：

```
void IMapServerToolAction.ServerAction(ToolEventArgs args) {
}
```

首先需要将屏幕坐标的线转换为地图坐标的线，代码如下：

```
Map map = args.Control as Map;
VectorEventArgs vectorargs = (VectorEventArgs)args;
Polyline mapPoly = ConvertHelper.GetMapPolyline(map, vectorargs);
```

上述代码调用了 ConvertHelper 类的 GetMapPolyline 方法，来实现转换功能。

在 App_Code 文件夹中新增加一 C#类，命名为 ConvertHelper。

在该类的头部加入如下命名空间的引用：

```
using ESRI.ArcGIS.ADF.Web.Geometry;
using ESRI.ArcGIS.ADF.Web.UI.WebControls;
```

然后加入转换线的 GetMapPolyline 方法。代码如下：

```
public static ESRI.ArcGIS.ADF.Web.Geometry.Polyline GetMapPolyline(
Map map, VectorEventArgs polyArgs) {
    System.Drawing.Point[] screenpoly = polyArgs.Vectors;

    PointCollection pc = new ESRI.ArcGIS.ADF.Web.Geometry.PointCollection();
    foreach (System.Drawing.Point dpnt in screenpoly) {
        pc.Add(Point.ToMapPoint(dpnt, map.Extent,
            (int)map.Width.Value, (int)map.Height.Value));
    }

    Path adf_path = new Path();
    adf_path.Points = pc;
    PathCollection adf paths = new PathCollection();
    adf paths.Add(adf path);
    Polyline mapPoly = new Polyline();
    mapPoly.Paths = adf paths;
    return mapPoly;
}
```

回到 LineBuffer 类的 ServerAction 方法中，然后利用如下代码将 Web ADF 的线对象转换为 ArcGIS Server 的线对象：

```
    ESRI.ArcGIS.ADF.ArcGISServer.PolylineN ags map polyline;
    ags map polyline =
        ESRI.ArcGIS.ADF.Web.DataSources.ArcGISServer.Converter.FromAdfGeometry(
mapPoly) as ESRI.ArcGIS.ADF.ArcGISServer.PolylineN;
```

由于缓冲区分析需要使用的是 COM 对象，因此需要先得到服务器上下文对象。代码如下：

```
    MapFunctionality mf;
    mf = (MapFunctionality)map.GetFunctionality("USAMap");
    if (mf == null)
        return;
    MapResourceLocal mrl;
    mrl = (MapResourceLocal)mf.MapResource;
    ESRI.ArcGIS.Server.IServerContext serverContext =
        mrl.ServerContextInfo.ServerContext;
```

得到服务器上下文对象后，便可以调用 BufferHelper 类的 Buffer 方法，对线对象进行缓冲区分析。代码如下：

```
    ESRI.ArcGIS.Geometry.IPolygon iBufferpolygon =
        BufferHelper.Buffer(serverContext, ags_map_polyline);
```

然后需要实现的是根据该缓冲区多边形，对美国各县进行空间查询，得到与该多边形相交的要素。代码如下：

```
    ESRI.ArcGIS.ADF.ArcGISServer.PolygonN[] queryResults =
        BufferHelper.Query(serverContext, bufferpolygon);
```

下面要完成的工作就是显示这些几何图形对象。代码如下：

```
    ESRI.ArcGIS.ADF.ArcGISServer.MapDescription mapDescription =
        mf.MapDescription;
    ESRI.ArcGIS.ADF.ArcGISServer.GraphicElement[] ges;
    ges =
       new ESRI.ArcGIS.ADF.ArcGISServer.GraphicElement[2 + queryResults.Length];

    ESRI.ArcGIS.ADF.ArcGISServer.SimpleFillSymbol querySym =
       AdfHelper.CreateSolidFillSymbol(255, 0, 0);

    for (int i = 0; i < queryResults.Length; i++) {
        ESRI.ArcGIS.ADF.ArcGISServer.PolygonElement queryPolygon;
        queryPolygon = new ESRI.ArcGIS.ADF.ArcGISServer.PolygonElement();
        queryPolygon.Symbol = querySym;
        queryPolygon.Polygon = queryResults[i];
        ges[i] = queryPolygon;
    }

    ESRI.ArcGIS.ADF.ArcGISServer.LineSymbol lnSymbol = null;
    lnSymbol = new ESRI.ArcGIS.ADF.ArcGISServer.SimpleLineSymbol();
```

```
    lnSymbol.Color = AdfHelper.CreateColor(0, 0, 0);
    lnSymbol.Width = 1;

    ESRI.ArcGIS.ADF.ArcGISServer.PolygonN buffer polyn;
    buffer polyn = ESRI.ArcGIS.ADF.Web.DataSources.ArcGISServer.Converter.Com
ObjectToValueObject(iBufferpolygon,serverContext,typeof(ESRI.ArcGIS.ADF.ArcGIS
Server.PolygonN)) as ESRI.ArcGIS.ADF.ArcGISServer.PolygonN;

    ESRI.ArcGIS.ADF.ArcGISServer.SimpleFillSymbol bufferSym =
        AdfHelper.CreateFillSymbol(255, 255, 0, lnSymbol);
    ESRI.ArcGIS.ADF.ArcGISServer.PolygonElement bufferPolyElement;
    bufferPolyElement = new ESRI.ArcGIS.ADF.ArcGISServer.PolygonElement();
    bufferPolyElement.Symbol = bufferSym;
    bufferPolyElement.Polygon = buffer polyn;

    ESRI.ArcGIS.ADF.ArcGISServer.LineElement drawPolyElement;
    drawPolyElement = new ESRI.ArcGIS.ADF.ArcGISServer.LineElement();
    drawPolyElement.Symbol = lnSymbol;
    drawPolyElement.Line = ags map polyline;

    ges[queryResults.Length] = bufferPolyElement;
    ges[queryResults.Length + 1] = drawPolyElement;

    mapDescription.CustomGraphics = ges;
    // 刷新地图
    map.Refresh();
```

编译并运行程序，通过线缓冲区工具，在地图上绘制一条线，双击结束画线，便可在地图上显示该线缓冲区以及该缓冲区影响的县。效果如图 7.16 所示。

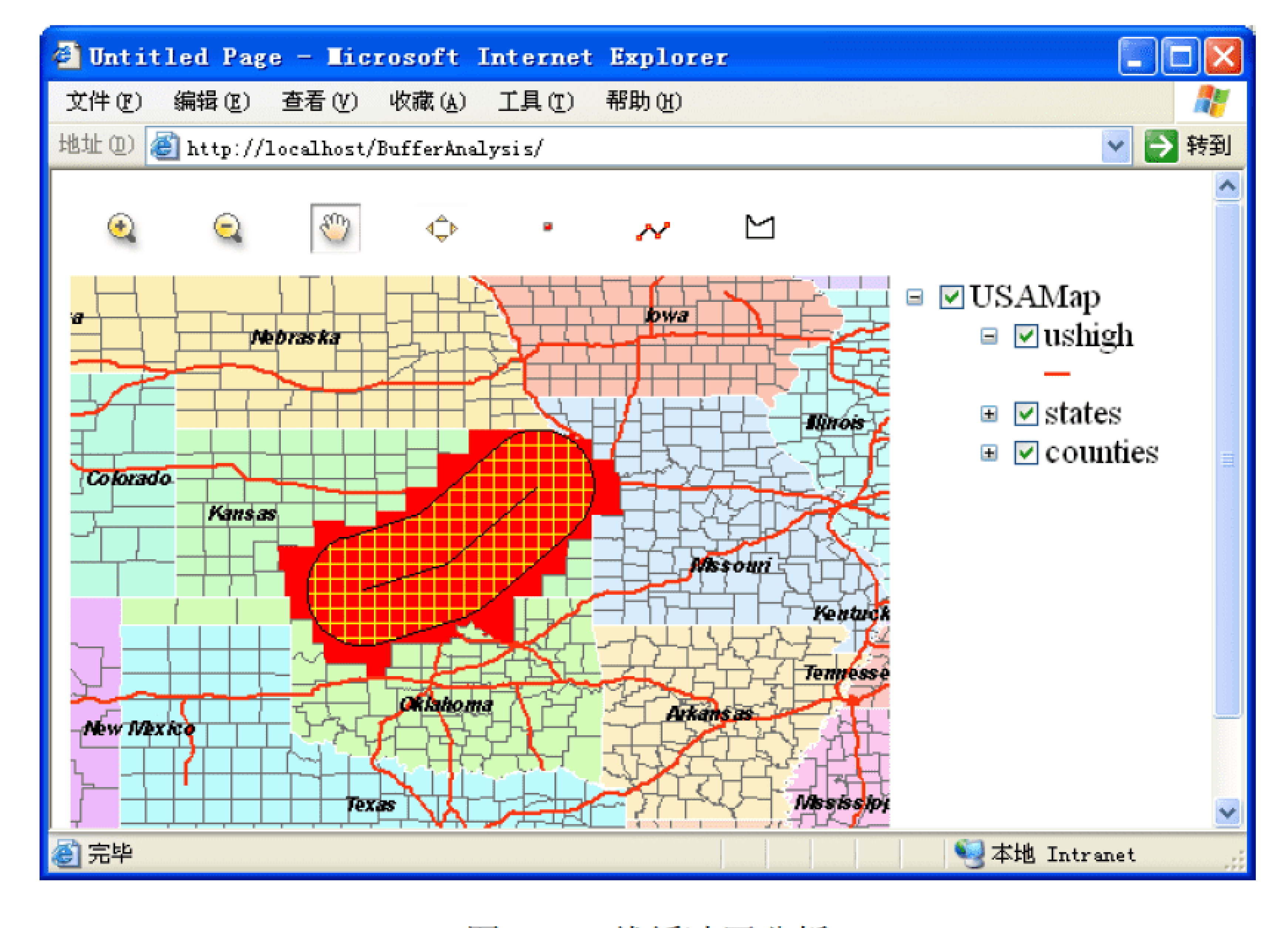


图 7.16 线缓冲区分析

3. 多边形缓冲区分析

在 App_Code 文件夹中新增加一 C#类，命名为 PolygonBuffer。

在类的头部加入如下命名空间的引用：

```
using ESRI.ArcGIS.ADF.Web.Geometry;
using ESRI.ArcGIS.ADF.Web.UI.WebControls;
using ESRI.ArcGIS.ADF.Web.UI.WebControls.Tools;
using ESRI.ArcGIS.ADF.Web.DataSources.ArcGISServer;
```

修改类的声明代码，增加对 IMapServerToolAction 接口的支持。代码如下：

```
public class PolygonBuffer: IMapServerToolAction
```

利用集成开发环境的代码自动生成功能，创建 IMapServerToolAction 接口的实现框架。

首先需要将屏幕坐标的多边形转换为地图坐标的多边形，代码如下：

```
Map map = args.Control as Map;
PolygonEventArgs polyArgs = (PolygonEventArgs)args;
Polygon mapPoly = ConvertHelper.GetMapPolygon(map, polyArgs);
```

上述代码调用了 ConvertHelper 类的 GetMapPolygon 方法，来实现转换功能。该方法的代码如下：

```
public static ESRI.ArcGIS.ADF.Web.Geometry.Polygon GetMapPolygon(Map map,
PolygonEventArgs polyArgs) {
    System.Drawing.Point[] screenpoly = polyArgs.Vectors;

    PointCollection pc = new ESRI.ArcGIS.ADF.Web.Geometry.PointCollection();
    foreach (System.Drawing.Point dpnt in screenpoly) {
        pc.Add(Point.ToMapPoint(dpnt, map.Extent,
            (int)map.Width.Value, (int)map.Height.Value));
    }

    Ring ring = new Ring();
    ring.Points = pc;
    RingCollection rings = new RingCollection();
    rings.Add(ring);
    Polygon mapPoly = new Polygon();
    mapPoly.Rings = rings;
    return mapPoly;
}
```

回到 PolygonBuffer 类的 ServerAction 方法中，然后利用如下代码将 Web ADF 的多边形对象转换为 ArcGIS Server 的多边形对象：

```
ESRI.ArcGIS.ADF.ArcGISServer.PolygonN ags map polygon;
ags map polygon =
    ESRI.ArcGIS.ADF.Web.DataSources.ArcGISServer.Converter.FromAdfGeometry(
mapPoly) as ESRI.ArcGIS.ADF.ArcGISServer.PolygonN;
```

由于缓冲区分析需要使用的是 COM 对象，因此需要先得到服务器上下文对象。代码如下：

```
    MapFunctionality mf;
    mf = (MapFunctionality)map.GetFunctionality("USAMap");
    if (mf == null)
        return;
    MapResourceLocal mrl;
    mrl = (MapResourceLocal)mf.MapResource;
    ESRI.ArcGIS.Server.IServerContext serverContext =
        mrl.ServerContextInfo.ServerContext;
```

得到服务器上下文对象后，便可以调用 BufferHelper 类的 Buffer 方法，对线对象进行缓冲区分析。代码如下：

```
    ESRI.ArcGIS.Geometry.IPolygon iBufferpolygon =
        BufferHelper.Buffer(serverContext, ags_map_polygon);
```

然后需要实现的是根据该缓冲区多边形，对美国各县进行空间查询，得到与该多边形相交的要素。代码如下：

```
    ESRI.ArcGIS.ADF.ArcGISServer.PolygonN[] queryResults =
        BufferHelper.Query(serverContext, bufferpolygon);
```

下面要完成的工作就是显示这些几何图形对象。代码如下：

```
    ESRI.ArcGIS.ADF.ArcGISServer.MapDescription mapDescription =
        mf.MapDescription;
    ESRI.ArcGIS.ADF.ArcGISServer.GraphicElement[] ges;
    ges =
        new ESRI.ArcGIS.ADF.ArcGISServer.GraphicElement[2 + queryResults.Length];

    ESRI.ArcGIS.ADF.ArcGISServer.SimpleFillSymbol querySym =
        AdfHelper.CreateSolidFillSymbol(255, 0, 0);

    for (int i = 0; i < queryResults.Length; i++) {
        ESRI.ArcGIS.ADF.ArcGISServer.PolygonElement queryPolygon;
        queryPolygon = new ESRI.ArcGIS.ADF.ArcGISServer.PolygonElement();
        queryPolygon.Symbol = querySym;
        queryPolygon.Polygon = queryResults[i];
        ges[i] = queryPolygon;
    }

    ESRI.ArcGIS.ADF.ArcGISServer.LineSymbol lnSymbol = null;
    lnSymbol = new ESRI.ArcGIS.ADF.ArcGISServer.SimpleLineSymbol();
    lnSymbol.Color = AdfHelper.CreateColor(0, 0, 0);
    lnSymbol.Width = 1;
    ESRI.ArcGIS.ADF.ArcGISServer.SimpleFillSymbol sfs1 =
        AdfHelper.CreateFillSymbol(255, 255, 0, lnSymbol);
    ESRI.ArcGIS.ADF.ArcGISServer.PolygonN buffer polyn;
    buffer polyn = ESRI.ArcGIS.ADF.Web.DataSources.ArcGISServer.Converter.Com
ObjectToValueObject(iBufferpolygon,serverContext,typeof(ESRI.ArcGIS.ADF.ArcGIS
Server.PolygonN)) as ESRI.ArcGIS.ADF.ArcGISServer.PolygonN;

    ESRI.ArcGIS.ADF.ArcGISServer.PolygonElement bufferPolyElement;
```

```
    bufferPolyElement = new ESRI.ArcGIS.ADF.ArcGISServer.PolygonElement();
    bufferPolyElement.Symbol = sfs1;
    bufferPolyElement.Polygon = buffer polyn;

    ESRI.ArcGIS.ADF.ArcGISServer.PolygonElement drawPolyElement;
    drawPolyElement = new ESRI.ArcGIS.ADF.ArcGISServer.PolygonElement();
    drawPolyElement.Symbol = sfs1;
    drawPolyElement.Polygon = ags_map_polygon;

    ges[queryResults.Length] = drawPolyElement;
    ges[queryResults.Length + 1] = bufferPolyElement;

    mapDescription.CustomGraphics = ges;
    // 刷新地图
    map.Refresh();
```

编译并运行程序，通过线缓冲区工具，在地图上绘制一多边形，双击结束画多边形，便可在地图上显示该多边形缓冲区以及该缓冲区影响的县。效果如图 7.17 所示。

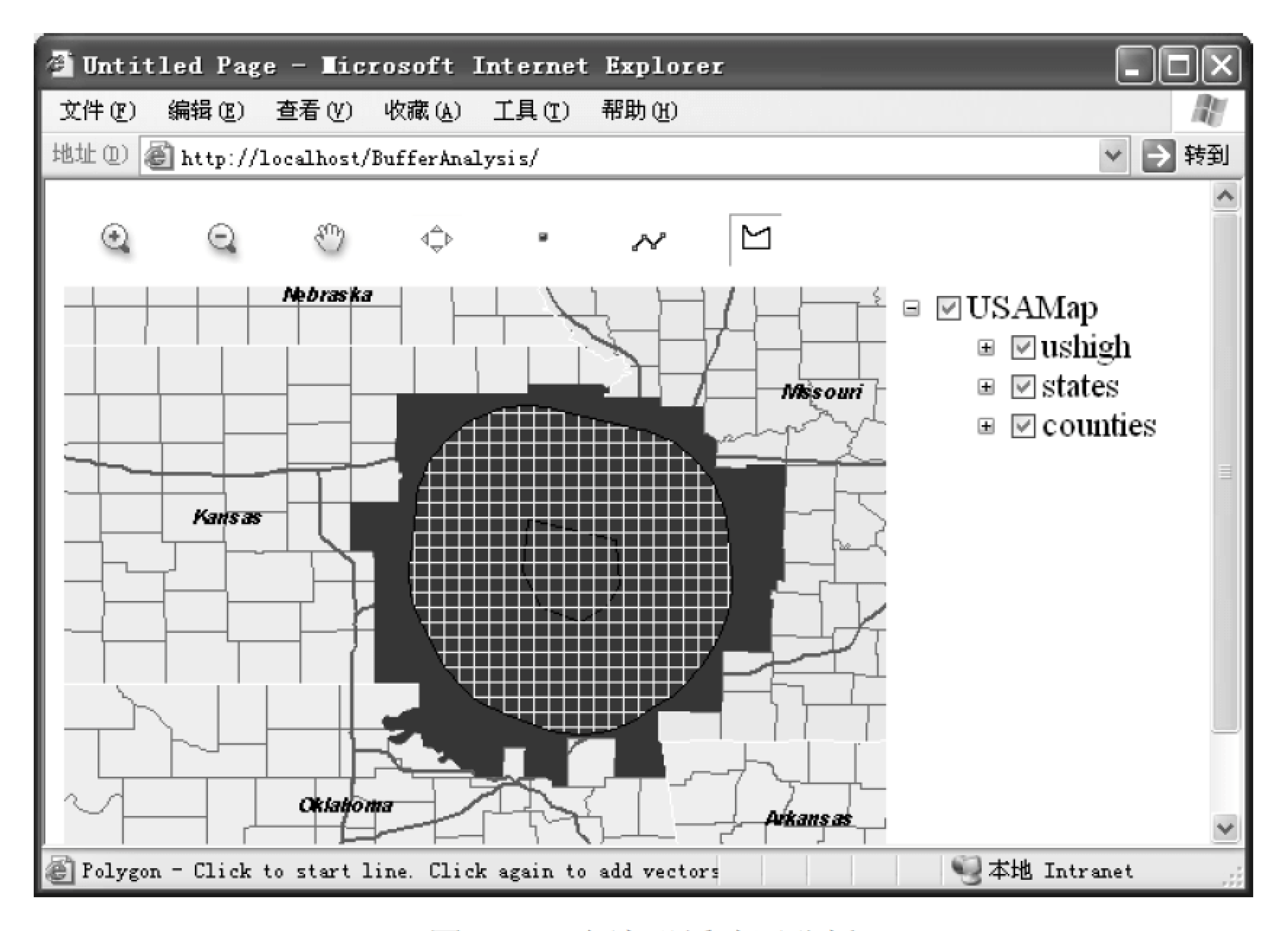


图 7.17　多边形缓冲区分析

第 8 章

任务框架

在 ESRI 产品的许多层次都存在任务的概念。任务就是提供公共结果的一组相关动作。例如 ArcGIS Server 空间处理服务，就是在 GIS 服务器执行空间分析的服务器任务。

Web ADF 提供了一个任务框架，用于创建、集成与分发组件（或称为 Web 任务）。实际上，Web 任务就是一 ASP.NET 的 Web 控件，该控件使用并扩展了 Web ADF 任务框架，在可分发的组件中封装了自定义功能。任务框架包含了一组创建自定义任务的接口与抽象类，一组支持任务管理与显示结果的控件，以及可集成到 Visual Stuido 与 ArcGIS Manager 中的功能。此外，Web ADF 中提供了现存的任务控件，用于支持在 Web 应用程序常用的 GIS 功能。

通过本章你将了解到：

8.1 任务控件

任务框架中包含了用于任务管理与接口显示的控件，例如 TaskManager 与 TaskResults，还包含了一组任务控件，例如 SearchAttributesTask、QueryAttributesTask、FindPlaceTask 等。

8.1.1 任务支持控件

任务支持控件包括 TaskManager 与 TaskResults。

TaskManager 控件用于在 Web 应用程序中组织与管理任务。TaskManager 控件生成 XML 结构的数据，这些数据可以在 ASP.NET 的导航控件中使用，例如 Menu 与 TreeView 控件。在程序运行期间，ASP.NET 的 Menu 或 TreeView 控件的节点可用于显示任务的浮动面板。因此该控件只在程序设计期间可见。

TaskResults 控件用于存储任务的结果。该结果可以用 TreeView 控件的节点来显示。TaskResults 控件也支持要素的放大、缩小与漫游、高亮显示结果集中的某一要素、重新执行某任务以及删除任务结果。

虽然并不一定需要在 Web 应用程序中使用任务支持控件，但通常可利用任务支持控件来提升 Web 任务用户体验。

8.1.2 Web ADF 提供的任务控件

Web ADF 提供了一组任务控件，这些控件实现并扩展了任务框架，可以完全集成到 Web 应用程序。

使用 SearchAttributesTask 控件，可根据用户输入的值，从资源中查询一组字段。在运行期间，查询每个字段，看是否包含用户提供的值。不过仅仅支持地图资源管理器中包含的资源的要素图层。查询结果的类型是 ADO.NET 的 DataSet，并显示在一 TaskResults 控件中。

通过 QueryAttributesTask 控件，则可指定某字段的查询参数。该控件提升了 SearchAttributesTask 控件的基本查询功能。在运行期间，一个 QueryAttributesTask 查询可提供一下拉列表框用于选择，或提供一文本框用于用户输入文本。每个查询可利用有效检查器来限制输入的值，也可以将几个查询组合在一块，得到一个结果。同样，该控件只支持地图资源管理器中包含的资源的要素图层。查询结果的类型也是 ADO.NET 的 DataSet，并显示在 TaskResults 控件中。

另外，FindPlaceTask 控件用于包含地名查询图层的 ArcWeb 服务。该控件包含一个用于输入地名的文本框。查询结果的类型是 ADO.NET 的 DataSet，并显示在 TaskResults 控件中。

FindAddressTask 控件使用地理编码资源管理器中的资源执行地址匹配操作。该控件将依据使用的资源，动态创建运行期间的界面。查询结果的类型是 ADO.NET 的 DataSet，并显示在 TaskResults 控件中。

GeoprocessingTask 控件使用空间处理资源管理器中的资源，执行空间处理任务。该控件将依

据空间处理资源要求的输入条件以及使用的任务，动态创建运行期间的界面。查询结果的类型是 ADO.NET 的 DataSet，并显示在一 TaskResults 控件中。

EditorTask 控件为 Web 应用程序提供一组编辑要素的功能。运行期间提供一对话框来修改、增加与删除要素的几何图形与属性信息。

8.1.3 任务框架的构成

图 8.1 显示了任务框架中包含的类与接口。

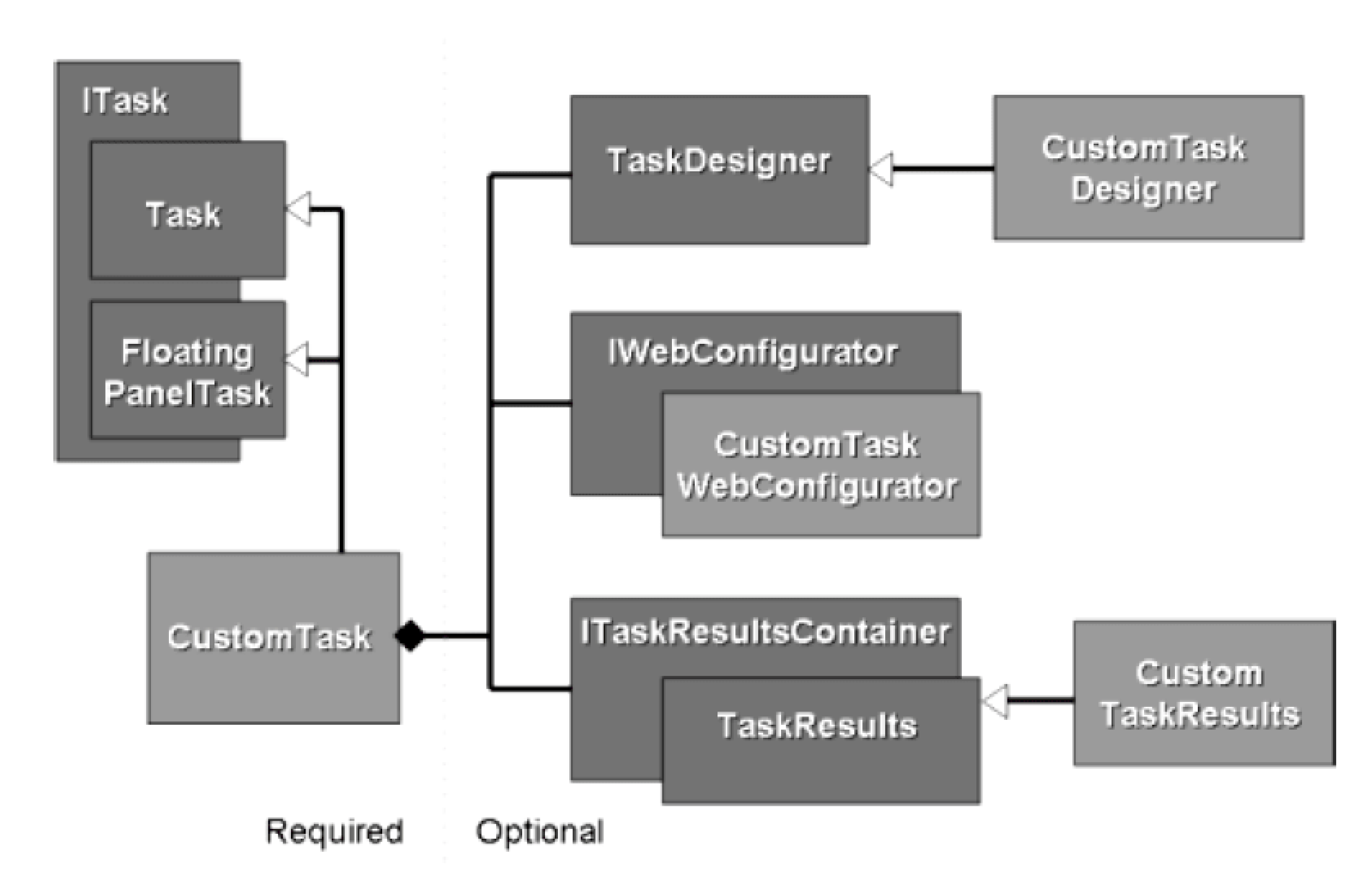


图 8.1 任务框架包含的类与接口

通常，Task 与 FloatingPanelTask 抽象类只实现了 ITask 接口中的基本方法。程序员可以根据任务的需求，必须继承这两个抽象类之一，创建自定义任务。但是，为了与其他 Web ADF 组件集成，程序员可以选择自定义其他任务组件。TaskDesigner 类允许程序员自定义程序设计期间的界面。IWebConfigurator 接口定义了如何将自定义任务集成到 Manager 应用程序中。这样在使用 Manager 创建 Web 应用程序时，便可选择该自定义的任务。ITaskResultsContainer 接口运行程序员开发一自定义的用户控件来显示任务结果。

任务框架中的核心接口是 ITask，位于 ESRI.ArcGIS.ADF.Web.UI.WebControls 命名空间中。所有的任务必须实现 ITask 接口。该接口的定义如下：

```
namespace ESRI.ArcGIS.ADF.Web.UI.WebControls {
    public interface ITask {
        string ShowUrl { get;}
        string Title { get; set;}
        string ToolTip { get;set;}
        string NavigationPath { get; set;}
        BuddyControlCollection TaskResultsContainers { get;}
        object Results { get;set;}
        void ExecuteTask();
        string TaskActivityIndicatorText { get;}
```

```
        bool GroupResultsByTable { get;set;}
        bool ShowFieldAttributes { get;set;}
        bool ShowLegend { get; set;}

        object Input { get; set;}
        void Refresh();
        void Show();
        CallbackResultCollection CallbackResults { get;}

        string UniqueID { get;}

        List<GISResourceItemDependency> GetGISResourceItemDependencies();
    }
}
```

Task 与 FloatingPanelTask 两个抽象类已经实现了 ITask 接口中最基础的功能，用户自定义任务时，可以从这两个类继承，作为起点。这两个类的最大不同在于：继承于 Task 的任务不能包含在 FloatingPanel 控件中，继承于 FloatingPanelTask 的任务在运行期间包含在一 FloatingPanel 控件中。

本质上任务就是一 ASP.NET 组合控件。因此，Task 与 FloatingPanelTask 都继承于 System.Web.UI.WebControls.CompositeControl，如图 8.2 所示。由于 CompositeControl 实现了 ICallbackEventHandler 接口，因此可以在 ASP.NET 的 Web 应用程序中应用 AJAX 功能。

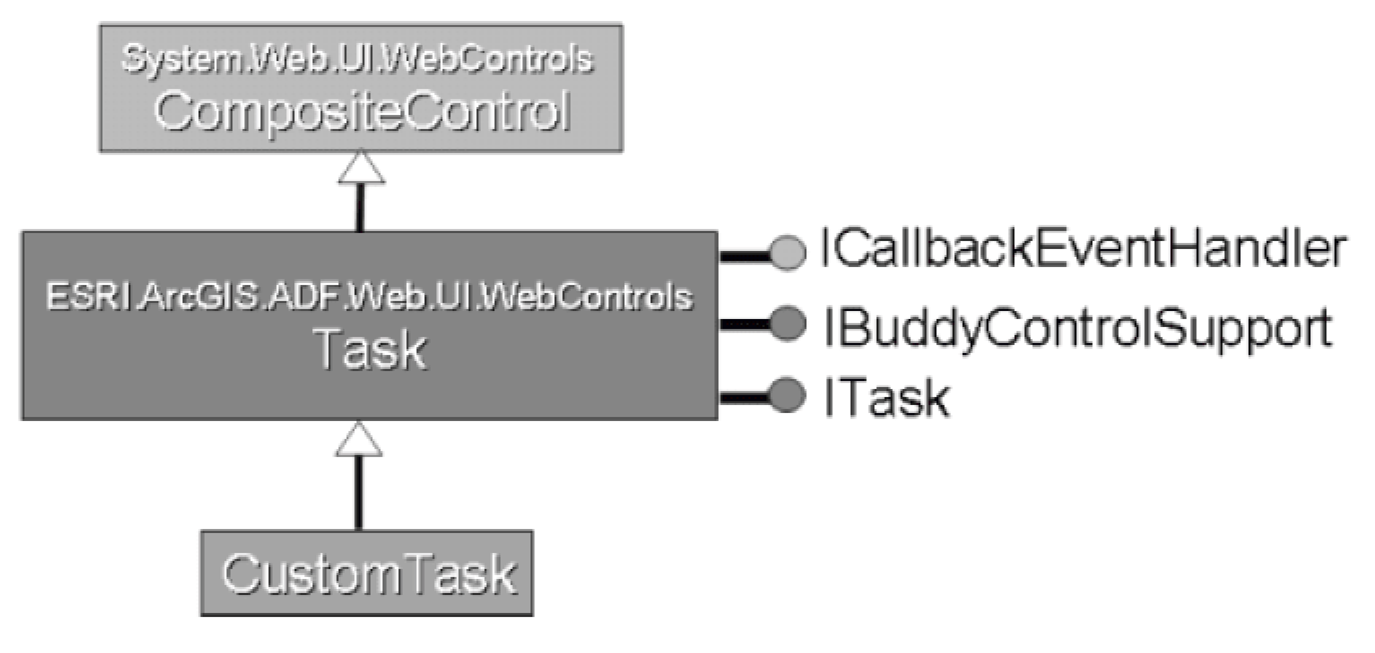


图 8.2　自定义任务需要继承的类

IBuddyControlSupport 接口定义了方法 GetSupportedBuddyControlTypes，来说明任务绑定的空间类型。如 OverviewMap 控件绑定 Map, Toolbar 控件绑定 Map 以及 PageLayout 控件。Task 与 FloatingPanelTask 类定义了任务能连接控件的类型。我们如果对 Map 操作，则可以使用如下代码：

```
public Type[] GetSupportedBuddyControlTypes() {
    return new Type[] { typeof(Map) };
}
```

8.2　自定义任务

由于使用 Web ADF 提供的任务控件方法很简单，并且已经在 4.1.2 节中简单做了介绍，因此我们不准备再详细介绍这些任务控件的使用。而是通过一个自定义任务的实例，来介绍整个任务框架。

本实例的任务在运行期间，包含一浮动面板，面板中包含一个按钮与一文本框。用户在文本框中输入，然后选择按钮，应用程序将根据文本框中的内容进行查询，并将结果显示在任务结果控件中。

8.2.1 自定义任务类库

新建一 Class Library 类型的工程，将工程名称设置为 QueryTask，解决方案名称命名为 CustomTask。

在工程中加入 System.Web.dll、System.Drawing.dll、ESRI.ArcGIS.ADF.Web.dll、ESRI.ArcGIS.ADF.Web.DataSource.dll 与 ESRI.ArcGIS.ADF.Web.UI.WebControls.dll 几个类库的引用。

将默认加入的 Class1.cs 文件重新命名为 SimpleQueryTask.cs。集成开发环境会自动将类名转换为 SimpleQueryTask。

在该类的头部加入如下命名空间的引用：

```
using System.Web.UI;
using System.Drawing;
using System.Web.UI.WebControls;
using System.Web.UI.HtmlControls;
using System.Collections.Specialized;
using ESRI.ArcGIS.ADF.Web.UI.WebControls;
using System.ComponentModel;
using System.Drawing.Design;
```

由于需要一个用户输入查询条件的界面，因此本自定义任务应该继承 FloatPanelTask 类。将 SimpleQueryTask 的声明代码修改为如下：

```
[ToolboxData("<{0}:SimpleQueryTask    runat=\"server\"    Width=\"100px\"
BorderWidth=\"1px\"> </{0}:SimpleQueryTask>")]
public class SimpleQueryTask : FloatingPanelTask
```

在上面的代码中，除了声明类的继承关系外，还利用 ToolboxData 属性类，指定当从 Microsoft Visual Studio 等工具中的工具箱拖动自定义控件时为它生成的默认标记。而其中出现的所有{0} 都将由设计器替换为与 SimpleQueryTask 类关联的标记前缀。

接着在类中加入一些字段，用于在运行期间引用其中的控件：

```
private TextBox textBox = null;
private TextBox phantomTextBox = null;
private HtmlInputButton button = null;
```

然后加入两个属性，ButtonText 与 Map。ButtonText 表示查询按钮显示的文本信息，而 Map 表示要查询的地图的 ID。代码如下：

```
[Browsable(true)]
[Category("Appearance")]
[DefaultValue("查询")]
```

```
    [PersistenceMode(PersistenceMode.Attribute)]
    public string ButtonText {
        get {
            object o = StateManager.GetProperty("buttonText");
            return (o == null) ? "Execute" : o as string;
        }
        set {
            StateManager.SetProperty("buttonText", value);
        }
    }

    /// <summary>
    /// 要查询的地图的 ID
    /// </summary>
    [Browsable(true)]
    [Category("SimpleQueryTask")]
    [DefaultValue("Map1")]
    [Description("要查询的地图的 ID，如果为空，则使用第一个地图控件。")]
    [PersistenceMode(PersistenceMode.Attribute)]
    public string Map {
        get {
            object o = StateManager.GetProperty("map");
            return (o == null) ? "Map1" : o as string;
        }
        set {
            StateManager.SetProperty("map", value);
        }
    }
```

覆盖（重载）父类的 CreateChildControls 方法。该方法在运行期间创建任务的可视界面，因此，输出结果必须可在浏览器中使用。该过程必须通过程序代码来实现，可使用.NET 的 HTML 控件类或 ASP.NET 控件的内容，来创建该界面。代码如下：

```
    protected override void CreateChildControls() {
        Controls.Clear();
        base.CreateChildControls();

        textBox = new TextBox();
        textBox.ID = "textBox";

        phantomTextBox = new TextBox();
        phantomTextBox.ID = "phantomTextBox";
        phantomTextBox.Style[HtmlTextWriterStyle.Display] = "none";

        button = new HtmlInputButton();
        button.ID = "button";
        button.Value = ButtonText;

        Controls.Add(textBox);
        Controls.Add(phantomTextBox);
```

```
        Controls.Add(button);

        string formateStr =
               "'textBoxValue=' + document.getElementById('{0}').value";
        string getArgumentJS = string.Format(formateStr, textBox.ClientID);
        string onClick = string.Format("executeTask({0},\"{1}\");",
                                       getArgumentJS,
                                       CallbackFunctionString);
        string onKeyDown = string.Format(
                         "if(event.keyCode==13){{{0}return false;}}", onClick);
        button.Attributes.Add("onclick", onClick);
        textBox.Attributes.Add("onkeydown", onKeyDown);
    }
```

在上述代码中，创建了包含两个单元格的 HTML 表格。其中一个单元格中包含一个文本框，另一个单元格包含一按钮。

覆盖父类的 GetCallbackResult 方法，在运行期间将 Input 属性设置为任务提交的值。Input 属性是在 FloatingPanelTask 类中定义的。该属性的类型是 object，但是通常仅仅就是一字符串。我们将在 ExecuteTask 方法中使用该属性。代码如下：

```
    public override string GetCallbackResult()
    {
        NameValueCollection keyValColl =
           CallbackUtility.ParseStringIntoNameValueCollection( callbackArg);
        Input = keyValColl["textBoxValue"];
        return base.GetCallbackResult();
    }
```

下面要实现的就是增加自定义任务的执行。代码如下：

```
    public override void ExecuteTask() {
        Results = null;
        if (Input == null)
           return;

        string textBoxValue = Input as string;

        Map mapCtrl = (Map)Utilities.FindControlRecursive(Page, this.Map);
        System.Data.DataSet ds = Utilities.AttributeQuery(mapCtrl, textBoxValue);

        Results = ds;
    }
```

在上面的代码中，通过 Input 属性，得到用户在文本框输入的查询条件，然后调用 Utilities 类的静态 AttributeQuery 方法执行查询。在调用该方法之前，调用了 Utilities 类的静态 FindControlRecursive 方法，从页面中查找到指定 ID 的地图控件。ExecuteTask 方法的代码最后将查询结果赋给 Results 属性。该属性的内容将显示在 TaskResults 控件中。TaskResults 类的 DisplayResult 方法负责处理 Results 对象。该对象的类型可以是 System.Data.DataSet 、ESRI.ArcGIS.ADF.Web.UI.WebControls.TaskResultNode 或 ESRI.ArcGIS.ADF.Web.UI.

WebControls.SimpleResultTask。（TaskResults 控件将以一树状结构来显示结果）

覆盖父类的 Refresh 方法。在运行期间处理在 TaskResults 控件中，重新运行任务与刷新任务。代码如下：

```
public override void Refresh() {
    string tmp = Input as string;
    if (!string.IsNullOrEmpty(tmp))
        textBox.Text = tmp;
    base.Refresh();
}
```

为完成自定义任务，还需要覆盖父类的 GetGISResourceItemDependencies 方法，该方法用于维护相关资源项集合。

```
public override List<GISResourceItemDependency>
GetGISResourceItemDependencies() {
    List<GISResourceItemDependency> list = new List<GISResourceItemDependency>();
    return list;
}
```

在当前工程中新增加名为 `Utilities` 的 C#类。在其头部加入如下命名空间的引用：

```
using System.Web.UI;
using System.Web.UI.WebControls;
using ESRI.ArcGIS.ADF.Web.UI.WebControls;
using ESRI.ArcGIS.ADF.Web.DataSources;
using ESRI.ArcGIS.ADF.Web;
```

FindControlRecursive 静态方法的代码如下，用于从指定控件中循环寻找指定 ID 的子控件：

```
public static Control FindControlRecursive(Control root, string id) {
if (root.ID == id) {
return root;
}

foreach (Control c in root.Controls) {
Control t = FindControlRecursive(c, id);
if (t != null) {
return t;
}
}

return null;
}
```

执行查询的 AttributeQuery 方法的代码如下。为了方便起见，我们直接在代码中指定查询 NorthAmericaMap 资源的 3 号图层，即美国州行政区划图层。

```
public static System.Data.DataSet AttributeQuery(Map map, string condition)
{
    IGISFunctionality gisfunc = map.GetFunctionality("NorthAmericaMap");
    if (gisfunc == null)
```

```
            return null;

        IGISResource gisresource = gisfunc.Resource;
        bool supportquery =
                gisresource.SupportsFunctionality(typeof(IQueryFunctionality));
        if (!supportquery)
            return null;

        IQueryFunctionality qfunc;
        qfunc = gisresource.CreateFunctionality(typeof(IQueryFunctionality), null)
as IQueryFunctionality;

        SpatialFilter spatialfilter = new SpatialFilter();
        spatialfilter.ReturnADFGeometries = false;
        spatialfilter.MaxRecords = 1000;
        spatialfilter.WhereClause = condition;

        System.Data.DataTable datatable = qfunc.Query(null, "3", spatialfilter);
        if (datatable == null)
            return null;

        System.Data.DataSet ds = new System.Data.DataSet();
        ds.Tables.Add(datatable);
        return ds;
    }
```

为了准备控件分发，需要为该任务指定一标记前缀。通常，该信息存储在 AssemblyInfo.cs 文件中。TagPrefix 属性定义在网页中用于标识自定义控件的标记前缀。

打开工程中 Properties 文件夹下的 AssemblyInfo.cs 文件，先加入如下命名空间的引用：

```
using System.Web.UI;
```

然后在该文件的最后加入如下代码，指定一标记前缀：

```
[assembly: TagPrefix("QueryTask", "queryTask")]
```

这样我们便初步完成了一个自定义任务。编译该自定义任务类库。

8.2.2 应用自定义的任务

通过 Visual Studio 2005 的 File 菜单的 Add New Web Site 命令，在当前解决方案中新增加一个站点，命名为 CustomTaskTest。

在 Default.aspx 页面中加入一地图资源管理器控件、一地图控件、一任务管理器控件与一任务结果控件。在地图资源管理器中加入 NorthAmericaMap 资源。

然后在工具箱中选择右键菜单的 Choose Items 命令，打开 Choose Toolbox Items 对话框。该对话框中，选择 Browse 按钮，选择 8.2.1 节创建的 QueryTask.dll 接口。选择 OK 保存设置。这时就会在工具箱中显示我们自定义的任务控件。将该控件拖到任务管理器控件中。

将自定义任务控件的相关控件设置为任务结果控件。

编译并运行程序，在自定义任务界面中输入查询条件，选择“查询”后，便可得到查询结果。运行效果如图 8.2 所示。

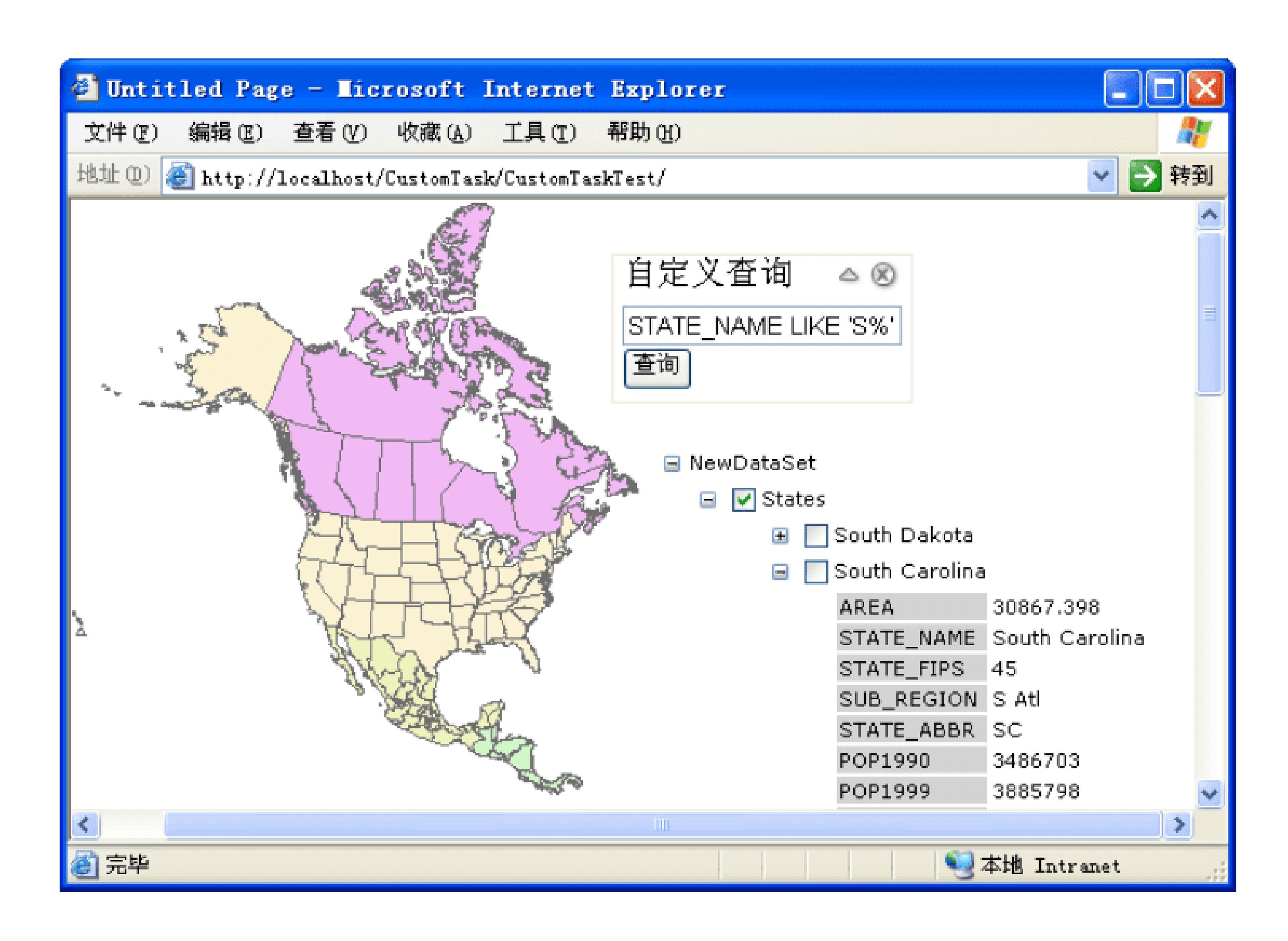


图 8.2　自定义任务运行效果

8.2.3　任务运行流程

在应用程序运行期间，大多数的任务遵循标准的流程，先处理用户请求，并生成与显示结果。Web ADF 的 JavaScript、Web ADF 控件类、任务框架接口及其实现共同支撑任务的工作。图 8.3 演示了本实例运行流程步骤。

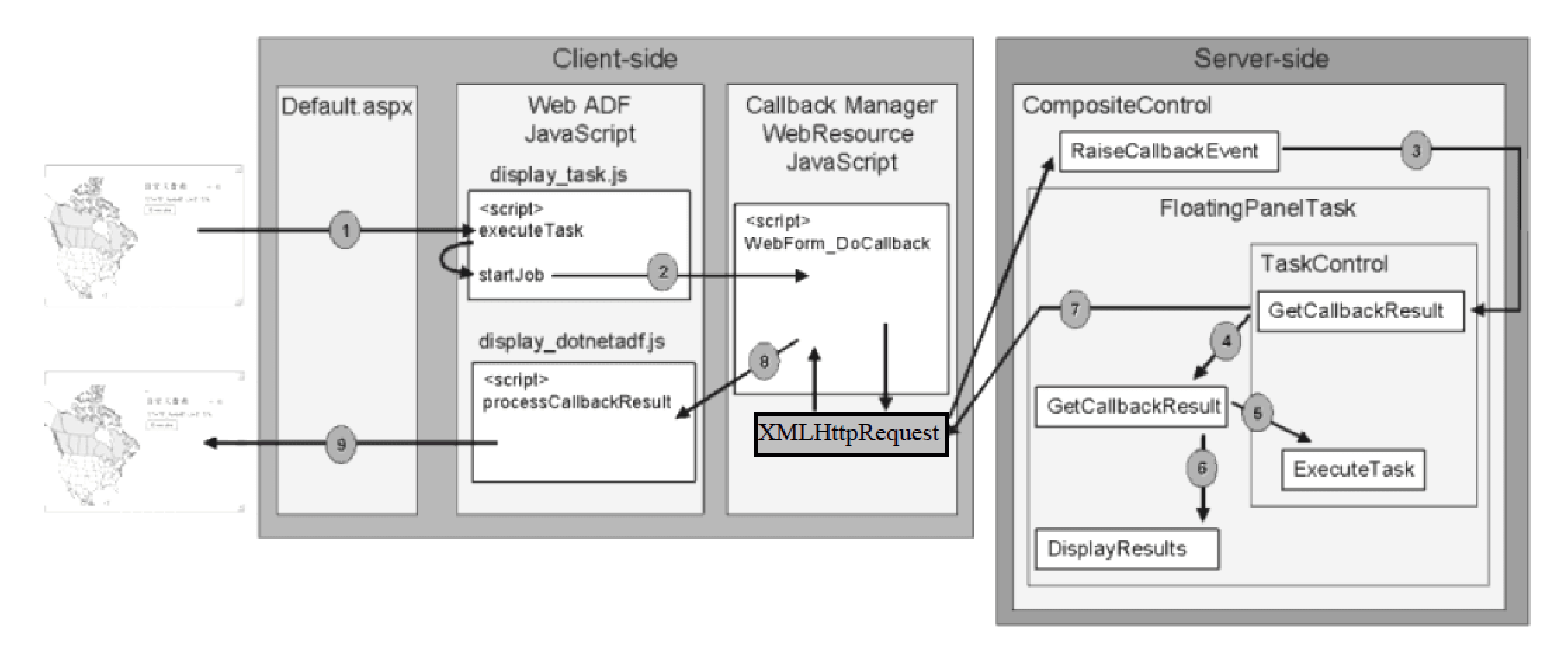


图 8.3　本实例自定义任务运行流程

本实例自定义任务运行流程步骤如下：

（1）用户在浏览器中单击按钮查询时，应用程序调用 Web ADF JavaScript display_task.js 文件

的 executeTask 函数。任务控件定义了用户动作。executeTask 函数中两个最重要的参数分别是回调参数与回调函数字符串。回调参数提供了用户操作的信息，例如提供查询数据库的字符串。回调函数字符串定义了 ASP.NET 回调管理器函数的内容，即 WebForm_DoCallback。读者可以通过网页右键菜单的“查看源文件”命令，查看按钮的 onclick 事件响应函数的代码，基本如下：

```
    onclick="executeTask('textBoxValue='+document.getElementById('TaskManager1
_SimpleQueryTask1_textBox').value',"WebForm_DoCallback('TaskManager1$  SimpleQu
eryTask1, argument, processCallbackResult, context, postBackError, true)");"
```

executeTask 函数在执行任务之前，完成两个工作，一是为异步请求指定一任务 ID，另一个是初始化一回调，启动在任务结果控件中的正在执行状态显示器。然后调用 startJob 函数处理用户的输入。

（2）startJob 函数创建任务处理用户请求需要的参数。任务控件使用 EventArg 参数来确定是否应该执行以及提供结果。自定义参数提供了用户在文本框输入的信息。这些值打包为一个字符串，并传递给 WebForm_DoCallback 函数的 argument 参数。例如在本实例中，使用的都是如下的参数，其中 textBoxValue 段是自定义参数：

```
    "EventArg=executeTask&taskJobID=2&textBoxValue=STATE_NAME LIKE 'S%'"
```

WebForm_DoCallback 函数中第一个参数表示处理回调请求的控件的名称，在本实例中，该名称应该为任务控件运行期间的名称。在 Web 端，XMLHttpRequest 对象初始化该回调，并将内容传递给任务控件。由于任务控件继承于 FloatingPanelTask，所以 RaiseCallbackEvent 方法将捕捉到该参数，并保存在_callbackArg 成员变量中。在执行任务时需要使用该成员变量。

（3）在回调阶段，RaiseCallbackEvent 执行完成之后，将调用 GetCallbackResult 方法。在任务控件中，利用该方法从自定义参数中获取用户输入的值。该值存储在 Input 属性中。在我们的实例中，使用 Web ADF 的 CallbackUtility 类处理_callbackArg 成员变量，将其处理为“名称\值”对。由于只有自定义参数，即 textBoxValue，有用，因此将其保存在 Input 属性中。

（4）FloatingPanelTask 类的 GetCallbackResult 方法负责调用执行任务的方法，以及构建在浏览器中显示的回调响应字符串。首先处理_callbackArg 变量，确定主要事件参数 EventArg。FloatingPanelTask 类寻找两个特定的值，分别是 startTaskActivityIndicator 与 executeTask。如果是 startTaskActivityIndicator，则在任务结果控件中显示一正在执行显示器。如果是 executeTask，在调用自定义任务类中的 ExecuteTask 方法，然后调用 FloatingPanelTask 类中的 DisplayResults 方法，创建一回调字符串。GetCallbackResult 方法的代码如下：

```
    public override string GetCallbackResult()
    {
       NameValueCollection keyValColl =
         CallbackUtility.ParseStringIntoNameValueCollection( callbackArg);

       string eventArg = keyValColl["EventArg"];
       string taskJobID = keyValColl["taskJobID"];

       if (eventArg == "startTaskActivityIndicator") {
          StartTaskActivityIndicator(taskJobID);
       }
```

```
        else if (eventArg == "executeTask") {
            ExecuteTask();
            DisplayResults(taskJobID, Input, Results);
        }
        else {
            return base.GetCallbackResult();
        }

        return CallbackResults.ToString();
    }
```

（5）自定义的任务类的 ExecuteTask 方法是任务处理任务的主体。在这里可以使用 Web ADF 的控件、资源、功能以及数据源特有的 API，处理保存在 Input 属性中的输入值。任务执行的结果保存在名为 Results 的属性中，该属性中的内容将显示在任务结果控件中。ITask 接口定义了 Results 属性，而 FloatingPanelTask 类实现了该接口，因此自定义任务类也有该属性。该属性可以是三种类型，分别是 SimpleTaskResult、DataSet 与 TaskResultNode。

（6）然后调用 FloatingPanelTask 类的 DisplayResults 方法。该方法将 Results 属性对象加入到任务结果控件中。任务结果控件本身有一个 DisplayResults 方法，用于在浏览器中显示 HTML 内容。Web ADF 控件使用 CallbackResult 对象更新该控件中的显示内容。当服务器端任务结果控件改变时，它创建一 CallbackResults 集合，而该集合用于创建传递给浏览器的回调响应。因此，需要将任务结果控件中的 CallbackResults 加入到自定义任务控件的 CallbackResults 属性中。FloatingPanelTask 控件的 DisplayResults 方法的代码如下：

```
    protected virtual void DisplayResults(string jobID, object taskInput, object
taskResults) {
        ITaskResultsContainer resultsContainer = null;
        for (int i = 0; i < TaskResultsContainers.Count; i++) {
            resultsContainer = Utility.FindControl(TaskResultsContainers[i].Name,
Page)
                        as ITaskResultsContainer;
            if (resultsContainer != null) {
                resultsContainer.DisplayResults(this,      jobID,      taskInput,
taskResults);
                CallbackResults.CopyFrom(resultsContainer.CallbackResults);
            }
        }
    }
```

（7）重新调用自定义任务控件的 GetCallbackResult 方法，将回调响应发送到客户端的浏览器中。自定义任务控件使用 CallbackResults 属性来存储需要返回给浏览器的内容字符串。Web ADF 的 JavaScript 函数处理该字符串，更新浏览器中 Web ADF 控件的显示。CallbackResults 属性存储了 CallbackResult 对象的集合。

（8）回调响应处理函数处理结果字符串。在本实例中，使用的是 Web ADF 定义的 processCallbackResult 函数。该函数用于解析回调响应，更新客户端的显示。

（9）根据回调响应结果更新浏览器中的控件。在本实例中，用查询结果更新任务结果控件。

8.2.4 自定义任务的改进

为了改善自定义任务控件在 Visual Studio 设计期间的属性的设置与修改，我们可以继承 ESRI.ArcGIS.ADF.Web.UI.WebControls.Design.Designers 命名空间中的 TaskDesigner 或 ControlTypeEditor。其中 ControlTypeEditor 用于提供一下拉列表框，让用户选择其中一个值来设置属性。

在 QueryTask 工程中，新增加一个类，命名 MapControlEditor。首先加入如下命名空间的引用：

```
using ESRI.ArcGIS.ADF.Web.UI.WebControls.Design;
```

然后将类的代码修改为如下：

```
class MapControlEditor : ControlTypeEditor {
    protected override void DefineSupportedTypes(
        System.ComponentModel.ITypeDescriptorContext pContext) {
        ClearSupportedTypes();
        AddSupportedType(typeof(ESRI.ArcGIS.ADF.Web.UI.WebControls.Map));
    }
}
```

上面的代码指定在下拉列表框中显示页面中所有地图控件的名称。

我们可以通过.NET 的属性（Attribute）来应用编辑器。

切换到 SimpleQueryTask.cs 文件中，在 Map 属性（Property）加入一编辑器属性，应用上述继承于 ControlTypeEditor 类的编辑器。Map 属性的代码最终如下所示：

```
[Browsable(true)]
[Category("SimpleQueryTask")]
[DefaultValue("Map1")]
[Description("要查询的地图的 ID，如果为空，则使用第一个地图控件。")]
[Editor(typeof(MapControlEditor), typeof(UITypeEditor))]
[PersistenceMode(PersistenceMode.Attribute)]
public string Map {
    get {
        object o = StateManager.GetProperty("map");
        return (o == null) ? "Map1" : o as string;
    }
    set {
        StateManager.SetProperty("map", value);
    }
}
```

在上述代码中，我们利用 Editor 属性指定用来更改 Map 属性的编辑器为 MapControlEditor。

重新编译 QueryTask 工程。并在 CustomTaskTest 工程中，重新加入该自定义任务控件。这时在该控件的属性编辑面板中，就可以通过下拉列表框来为 Map 属性设置值，效果如图 8.4 所示。

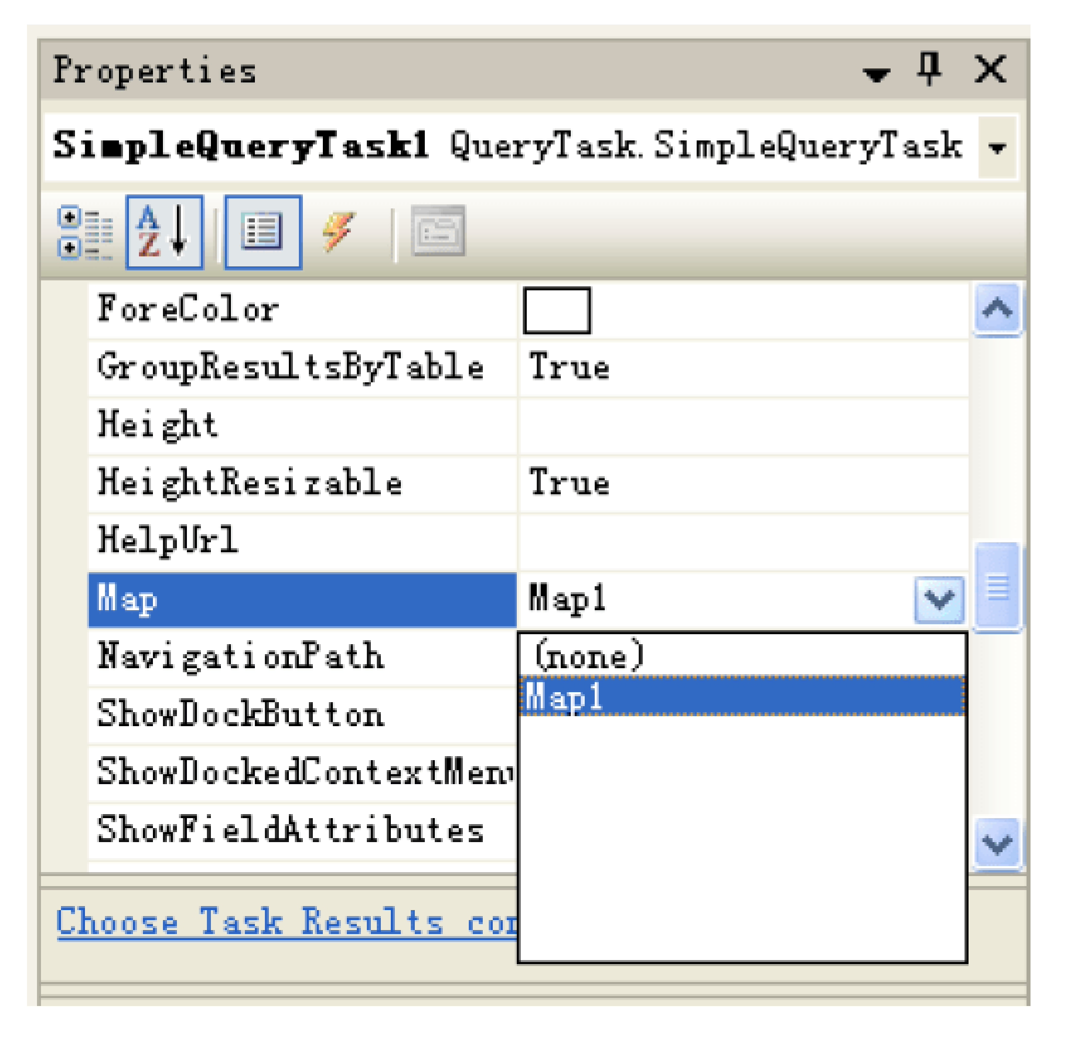


图 8.4　应用自定义编辑器

第 9 章

扩展ArcGIS服务器

开发人员可以使用服务器API开发任何类型的应用程序，开发ArcGIS Server应用程序时，处理的是运行在GIS服务器上的远程ArcObjects对象。因此，服务器API包含了许多ArcObjects组件，这些组件包含在一组对象库中。在本章中，将介绍如何通过扩展ArcGIS服务器提高应用程序的性能。

9.1　几个概念

在扩展 GIS 服务器之间，先来认识几个相关的概念。

9.1.1　ArcGIS 服务器与细粒度的 ArcObjects

ArcGIS 服务器同 ArcGIS Engine 与 ArcGIS 桌面应用程序一样，使用同一套 ArcObjects，因此，如果在服务器上充分利用 ArcObjects，那么就可以创建与 ArcGIS 桌面应用程序类似功能与性能。

在应用程序中，通过 ArcGIS 服务器 API，既可以使用粗粒度的远程 ArcObjects，例如 MapServer、GeoCodeServer、GeoDataServer、GPServer 与 GlobeServer 以及它们的扩展的方法，也可以使用细粒度的远程 ArcObjects，例如循环处理一多边形中的结点。但是值得注意的是，在应用程序中调用运行在服务器上的对象，属于跨进程调用。例如，对于一 Web 应用程序，Web 应用程序运行在一进程中，而远程对象运行在另一进程中。

跨进程调用的速度要比同进程中调用慢许多。通常还有可能的是 Web 应用程序与对象运行在不同的计算机上，因此远程调用不仅跨进程，而且跨计算机。单独的一次调用，或者是几十次的调用，对于应用程序的整体性能影响意义不明显。但是，如果应用程序上千次地跨进程或跨计算机调用 GIS 服务器上的细粒度的 ArcObjects，那么肯定会对应用程序的性能有明显的影响。

因此，开发人员应该通过最小化细粒度调用远程对象的次数，来最小化应用程序与 GIS 服务器的往返交互。如果在应用程序中，确实需要频繁调用细粒度的 ArcObjects，可以使用两种策略来支持这种功能，同时保持远程调用的最小化。一个策略是通过利用 COM 对象的应用程序扩展 GIS 服务器，另一个是使用自定义的服务器对象扩展。每个策略都各有优势。

使用 COM 对象来扩展 GIS 服务器很简单，可允许开发人员在 ArcGIS 服务器、ArcGIS 桌面产品以及 ArcGIS Engine 之间共享自定义的控件。这些 COM 功能对象不需要与任何特定的服务器对象配置或类型绑定，甚至可以在一个空的服务器上下文中使用，如图 9.1 所示。但是这种方法有一定的限制。当使用 COM 功能对象扩展 GIS 服务器时，对于服务器对象或服务器上下文每次请求，都会创建一新的对象，这意味着对于池化的服务器对象，对服务器对象的每个请求，必须创建该 COM 功能对象。如果该 COM 对象初始化时需要消耗很高的资源，那么应该禁止这种持续的创建。此外，由于每次请求都创建新的 COM 对象，因此不能使用该对象来缓存它使用的信息。

服务器对象扩展与 COM 功能对象正相反。它与服务器对象本身一样，在服务器对象初始化时，创建与初始化服务器对象扩展，并在请求级别重复使用。因此，即使服务器对象扩展初始化需要消耗大量资源，那么该消耗也只有在服务器对象创建时才出现，是一次性的。同时，由于只要服务器对象存在，服务器对象扩展实例就存在，因此可以在请求之间重复使用它的缓存信息。但是，服务器对象扩展与某特定的服务器对象配置或类型绑定，不能像 COM 功能对象那样广泛使用。

如果应用程序使用细粒度的 ArcObjects，也不是总是必须扩展服务器或服务器对象。正如上面介绍的，这决定于应用程序调用服务器的数量。如果应用程序中常常需要上千次地调用细粒度的 ArcObjects，那么就应用考虑将其中一些代码移动到 GIS 服务器端。

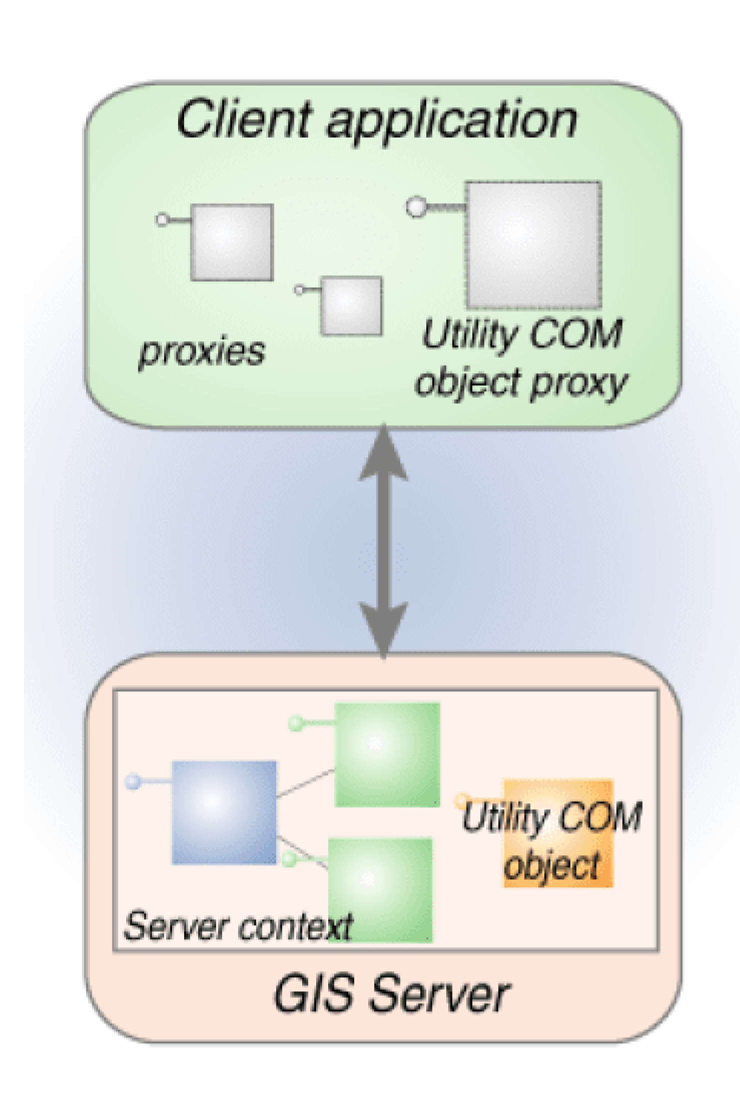


图 9.1 使用 COM 功能对象扩展 GIS 服务器

9.1.2 GIS 服务器 COM 对象与.NET

可以使用 COM 语言，例如 VB、C++或.NET，来开发 COM 功能对象。而服务器对象扩展也是作为 COM 对象实现的，可以使用 C++或.NET 来开发。当使用.NET 创建 GIS 服务器使用的 COM 对象时，应该遵循如下一些规则：

（1）必须显式创建 COM 类实现的接口。与 Visual Basic 6 不同，.NET 不会自动为 COM 类创建一个显式接口，而在服务器上下文创建对象时需要使用该接口。

（2）COM 类应该生成一双重类接口。这可以通过在类上增加 ClassInterface 属性，并将其值指定为 ClassInterfaceType.AutoDual 来实现。

（3）确保 COM 对象在 GIS 服务器上能正确执行。COM 对象必须继承于 ServicedComponet 类，该类位于 System.EnterpriseServerices 程序集中。

9.2 使用 COM 功能对象扩展 GIS 服务器

扩展 GIS 服务器的 COM 功能对象必须安装在服务器对象容器（SOC）计算机上，只有这样应用程序才能访问。例如，我们自定义了一个网络分析功能，希望能在 GIS 服务器上使用，可以使用如下代码调用 COM 对象：

```
mylib.TraceUtilities tracer =
    pServerContext.CreateObject("mylib.TraceUtilities")
    as mylib.TraceUtilities;
result = tracer.DoIsolationTrace(...);
```

在上面的代码中，分析功能可能包含对 ArcObjects 的上千次的调用，但是所有这些调用都在

TraceUtilities 这个 COM 对象运行的 GIS 服务器上执行。粗粒度的 DoIsolationTrace 方法是 Web 应用程序唯一需要调用的，因此这意味着只需要对 GIS 服务器的一次远程调用。

下面我们通过一个实例来介绍如何利用 COM 功能对象来扩展 GIS 服务器。该实例包括两部分。一部分是功能组件，另一个部分是 Web 应用程序，该程序将利用 ArcGIS Server ArcObjects API 来访问功能组件提供的自定义功能。

9.2.1 COM 组件的创建与实现

在 Visual Studio 中，通过 File 菜单的 New Project 命令，创建一类型为 Class Library 的功能，将工程命名为 VegCOM，将解决方案命名为 SpatialQueryCOM。

在工程中加入 ESRI.ArcGIS.Carto、ESRI.ArcGIS.Catalog、ESRI.ArcGIS.CatalogUI、ESRI.ArcGIS.Display、ESRI.ArcGIS.Framework、ESRI.ArcGIS.Geodatabase、ESRI.ArcGIS.Geometry、ESRI.ArcGIS.Server、ESRI.ArcGIS.esriSystem 以及 System.EnterpriseServices 程序集的引用。

在工程中加入一个名为 IVegResults 的接口。在其文件的头部加入如下命名空间的引用：

```
using ESRI.ArcGIS.Carto;
using ESRI.ArcGIS.Geodatabase;
using System.Runtime.InteropServices;
```

然后在接口中加入两个属性，一个代表一组图形元素，另一个代表记录集。这两个属性的代码如下：

```
IGraphicElements ResGraphics {
    get;
    set;
}

IRecordSet Stats {
    get;
    set;
}
```

由于这里使用的是 COM 接口，需要指定接口的 GUID。这可以通过在接口定义处增加 GuidAttribute 属性来完成。即在接口声明代码前一行加入如下代码：

```
[GuidAttribute("")]
```

可使用 GUID 生成工具生成一个 GUID。Visual Studio 2005 安装时，附带安装了一个 GUID 生成工具 guidgen.exe，位于 Visual Studio 安装路径的 Comcom7\Tools 目录下。通过 Tools 菜单的 External Tools 命令，打开如图 9.2 所示的 External Tools 对话框。可以通过该对话框将 GUID 生成工具命令加入到 Tools 菜单中。

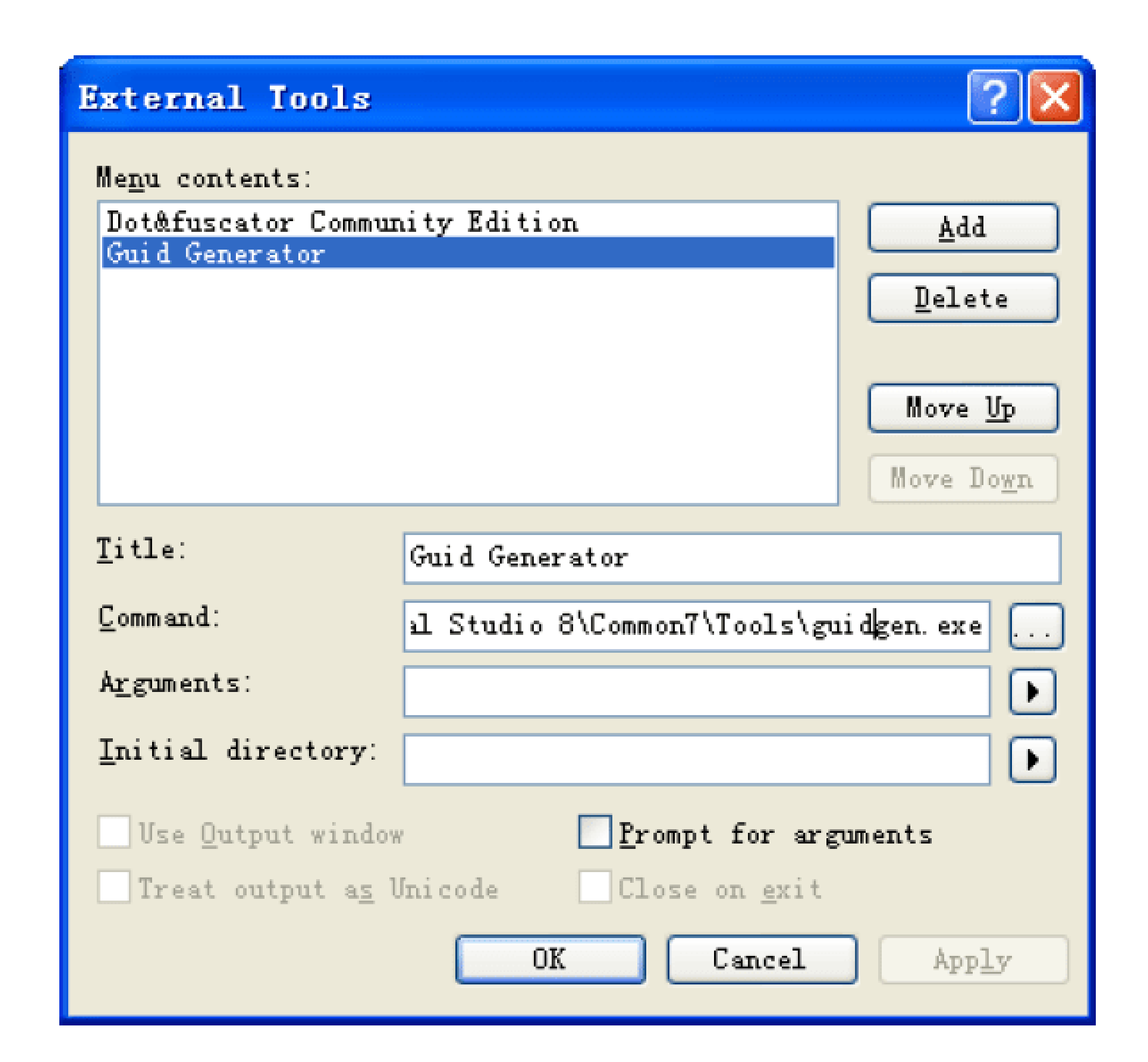


图 9.2 通过 External Tools 对话框维护 Tools 菜单中的外部命令

然后通过该命令，打开如图 9.3 所示的 Create GUID 对话框。在该对话框中，指定 GUID 的格式为 Registry Format，选择 New GUID 按钮，生成一个新的 GUID，然后选择 Copy 按钮复制该 GUID。将该 GUID 粘贴到属性中，并删除其中的“｛”与“｝”。

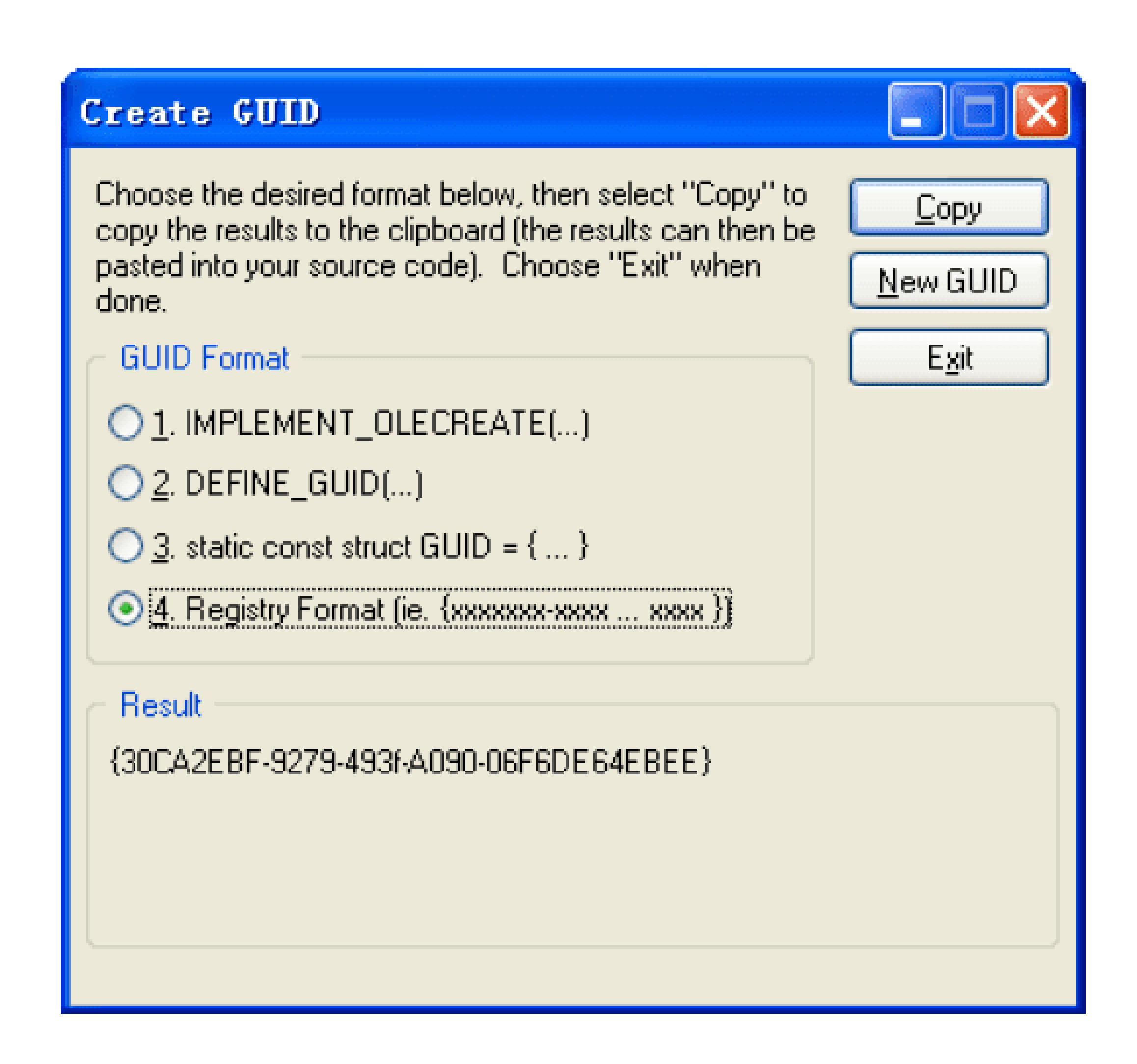


图 9.3 Create GUID 对话框

在工程中新增加名为 VegResults 的类。在类的头部加入如下命名空间的引用：

```
using System.Runtime.InteropServices;
using System.EnterpriseServices;
using ESRI.ArcGIS.Carto;
using ESRI.ArcGIS.Geodatabase;
```

该类也同样需要一些类的属性，特别是 AutomationProxy、ClassInterface 与 GuidAttribute。在类定义的前一行加入如下代码：

```
[AutomationProxy(true),ClassInterface(ClassInterfaceType.AutoDual),
GuidAttribute("")]
```

使用 GUID 生成工具为类创建另一个 GUID。AutomationProxy 属性指定是否应该使用自动化封送拆收器或自定义代理及存根（Stub）对该类型进行封送处理。ClassInterface 属性为公开给 COM 的类指定要生成的类接口的类型。

该类需要继承 ServicedComponent 类与实现 IVegResults 接口。将类的声明代码修改如下：

```
public class VegResults : ServicedComponent, IVegResults
```

在类中加入两字段，代码如下：

```
private IGraphicElements m_resGraphics;
private IRecordSet m_resStats;
```

然后加入两个属性，代码如下：

```
public IGraphicElements ResGraphics {
    get {
        return m_resGraphics;
    }
    set {
        m_resGraphics = value as IGraphicElements;
    }
}

public IRecordSet Stats {
    get {
        return m_resStats;
    }
    set {
        m_resStats = value as IRecordSet;
    }
}
```

上述代码实现了结果类，下面要实现的是 COM 功能类。

在工程中新增加一个名为 IVegUtils 的接口，该接口的代码如下：

```
using System;
using System.Runtime.InteropServices;
using ESRI.ArcGIS.Geodatabase;
using ESRI.ArcGIS.Geometry;

namespace VegCOM {
    [GuidAttribute("D3FE0F0A-245E-4e6c-92BC-005B9DC3D93A")]
    public interface IVegUtils {
        IVegResults sumVegetationType(ref IFeatureClass pVegClass,
            ref IPoint pPoint, ref double dDistance,
            ref string sSummaryFld);
    }
}
```

接口中包含一个名为 sumVegetationType 的方法，该方法用于以 pPoint 参数为圆心，以 dDistance 参数为半径，以 sSummaryFld 为统计字段，对 pVegClass 指定的图层统计每个类别的面积。

在工程中新增加一名为 VegUtils 的类。引用命名空间的代码如下：

```
using System.EnterpriseServices;
using System.Collections;
using System.Collections.Specialized;
using System.Runtime.InteropServices;
using ESRI.ArcGIS.Carto;
using ESRI.ArcGIS.Display;
using ESRI.ArcGIS.esriSystem;
using ESRI.ArcGIS.Geodatabase;
using ESRI.ArcGIS.Geometry;
```

类声明的代码如下：

```
[AutomationProxy(true)]
[ClassInterface(ClassInterfaceType.AutoDual)]
[GuidAttribute("1EBD03A5-1216-459f-845F-3A61749FAEF7")]
public class VegUtils : ServicedComponent, IVegUtils
```

类的 sumVegetationType 方法的代码如下：

```
public IVegResults sumVegetationType(ref IFeatureClass pVegClass,
    ref IPoint pPoint, ref double dDistance, ref string sSummaryFld) {
    // 计算点的缓冲区
    ITopologicalOperator pTopoOp = pPoint as ITopologicalOperator;
    IGeometry pGeom = pTopoOp.Buffer(dDistance);

    // 用缓冲区查询要素图层
    ISpatialFilter pSFilter = new SpatialFilter();
    pSFilter.Geometry = pGeom;
    pSFilter.SpatialRel = esriSpatialRelEnum.esriSpatialRelIntersects;
    pSFilter.GeometryField = pVegClass.ShapeFieldName;

    IFeatureCursor pFCursor = pVegClass.Search(pSFilter, true);

    // 对选择出来的要素进行循环，将每个几何类型裁剪到临时变量中
    pTopoOp = pGeom as ITopologicalOperator;
    int lPrim = pVegClass.FindField(sSummaryFld);

    ListDictionary dict = new ListDictionary();

    // 为几何图形创建符号与图形要素
    ISimpleFillSymbol pSFS = newFillS();
    IGraphicElements pGraphics = new GraphicElements();

    IFeature pFeature;
    while ((pFeature = pFCursor.NextFeature()) != null) {
        // 创建图形
        IFillShapeElement pFE = new PolygonElement() as IFillShapeElement;
```

```
        IElement pElement = pFE as IElement;

        // 裁剪几何类型
        IGeometry pNewGeom = pTopoOp.Intersect(pFeature.Shape,
                              esriGeometryDimension.esriGeometry2Dimension);
        pElement.Geometry = pNewGeom;
        pFE.Symbol = pSFS;
        IGraphicElement ge = pFE as IGraphicElement;
        pGraphics.Add(ge);

        // 加入到单链接列表对象中
        IArea pArea = pNewGeom as IArea;
        string sType = pFeature.get Value(lPrim) as string;
        if (dict.Contains(sType))
           dict[sType] = (double)dict[sType] + pArea.Area;
        else
           dict[sType] = pArea.Area;
    }

    // 创建汇总记录集
    IRecordSet psumRS = sumRS(dict);

    // 创建结果对象
    IVegResults pRes = new VegResults();
    pRes.ResGraphics = pGraphics;
    pRes.Stats = psumRS;

    return pRes;
}
```

该方法调用了 sumRS 方法来统计每个类别的面积。sumRS 方法的代码如下：

```
private IRecordSet sumRS(ListDictionary dict) {
    // 新创建记录集对象 create the new record set
    IRecordSet pNewRs = new RecordSet();
    IRecordSetInit prsInit = pNewRs as IRecordSetInit;

    IFields pFields = new Fields();
    IFieldsEdit pFieldsEdit = pFields as IFieldsEdit;
    pFieldsEdit.FieldCount 2 = 2;

    IField pField = new Field();
    IFieldEdit pFieldEdit = pField as IFieldEdit;
    pFieldEdit.Name 2 = "Type";
    pFieldEdit.Type 2 = esriFieldType.esriFieldTypeString;
    pFieldEdit.Length 2 = 50;
    pFieldsEdit.set Field(0, pField);

    pField = new Field();
    pFieldEdit = pField as IFieldEdit;
```

```
    pFieldEdit.Name_2 = "Area";
    pFieldEdit.Type_2 = esriFieldType.esriFieldTypeDouble;
    pFieldsEdit.set_Field(1, pField);

    prsInit.CreateTable(pFields);

    // 增加所有的类型/面积对
    ICursor pIC = prsInit.Insert();
    IRowBuffer pRowBuf = prsInit.CreateRowBuffer();

    IDictionaryEnumerator myEnumerator = dict.GetEnumerator();
    while (myEnumerator.MoveNext()) {
        pRowBuf.set_Value(0, myEnumerator.Key);
        pRowBuf.set_Value(1, myEnumerator.Value);
        pIC.InsertRow(pRowBuf);
    }

    return pNewRs;
}
```

创建符号的 newFillS 方法的代码如下：

```
private ISimpleFillSymbol newFillS() {
    ISimpleLineSymbol pSLS = new SimpleLineSymbol();
    IRgbColor pcolor = new RgbColor();
    pcolor.Red = 255;
    pcolor.Green = 0;
    pcolor.Blue = 0;
    pSLS.Color = pcolor;
    pSLS.Style = esriSimpleLineStyle.esriSLSSolid;
    pSLS.Width = 2;

    ISimpleFillSymbol pSFS = new SimpleFillSymbol();
    pSFS.Outline = pSLS;
    pSFS.Style = esriSimpleFillStyle.esriSFSHollow;

    return pSFS;
}
```

至此实现了 COM 组件，下面要做的是设置工程的程序集属性。打开 AssemblyInfo.cs 文件，将 ComVisibleAttribute 属性的值由原来的 false 修改为 true。该属性控制程序集中个别托管类型、成员或所有类型对 COM 的可访问性。当创建 Class Library 类型的工程时，集成开发环境默认将该值设置为 false，隐藏该程序集中的所有 public 类型。即将：

```
[assembly: ComVisible(false)]
```

修改为：

```
[assembly: ComVisible(true)]
```

9.2.2　注册 COM 组件

编译工程，将在 bin 目录下生成 VegCOM.dll 文件。

打开 Visual Studio 2005 命令行工具（开始 | 程序 | Mircrosoft .NET Framework 2.0 | SDK command prompt），切换到 VegCOM.dll 文件所在的目录。然后运行如下命令：

```
regasm VegCOM.dll /tlb:VegCOM.tlb /codebase
```

regasm 是一个程序集注册工具，用于读取程序集中的元数据，并将所需的项添加到注册表中。注册表允许 COM 客户程序以透明方式创建 .NET Framework 类。类一经注册，任何 COM 客户程序都可以使用它，就好像该类是一个 COM 类。类仅在安装程序集时注册一次。程序集中的类实例直到被实际注册时，才能从 COM 中创建。

9.2.3　使用 COM 功能对象

用 Manager 工具将 ArcGIS 安装目录下 DeveloperKit\SamplesNET\Server\data\Yellowstone 的 Yellowstone.mxd 发布为 Yellowstone 地图服务。

在解决方案中加入名为 COMTest 的 Web 站点。在 Default.aspx 页面中加入一地图资源管理器控件、一地图控件、一工具栏控件、一 Toc 控件、一标签控件、一文本框控件、一复选框控件、一 IMG 控件以及一个 Div 控件，并在 Div 控件中加入一个 GridView 控件。

在地图资源管理器控件中加入 Yellowstone 地图资源。在工具栏控件中加入“放大”、“缩小”、“漫游”工具与一个“自定义”的工具。页面在设计期间的视图效果如图 9.4 所示。

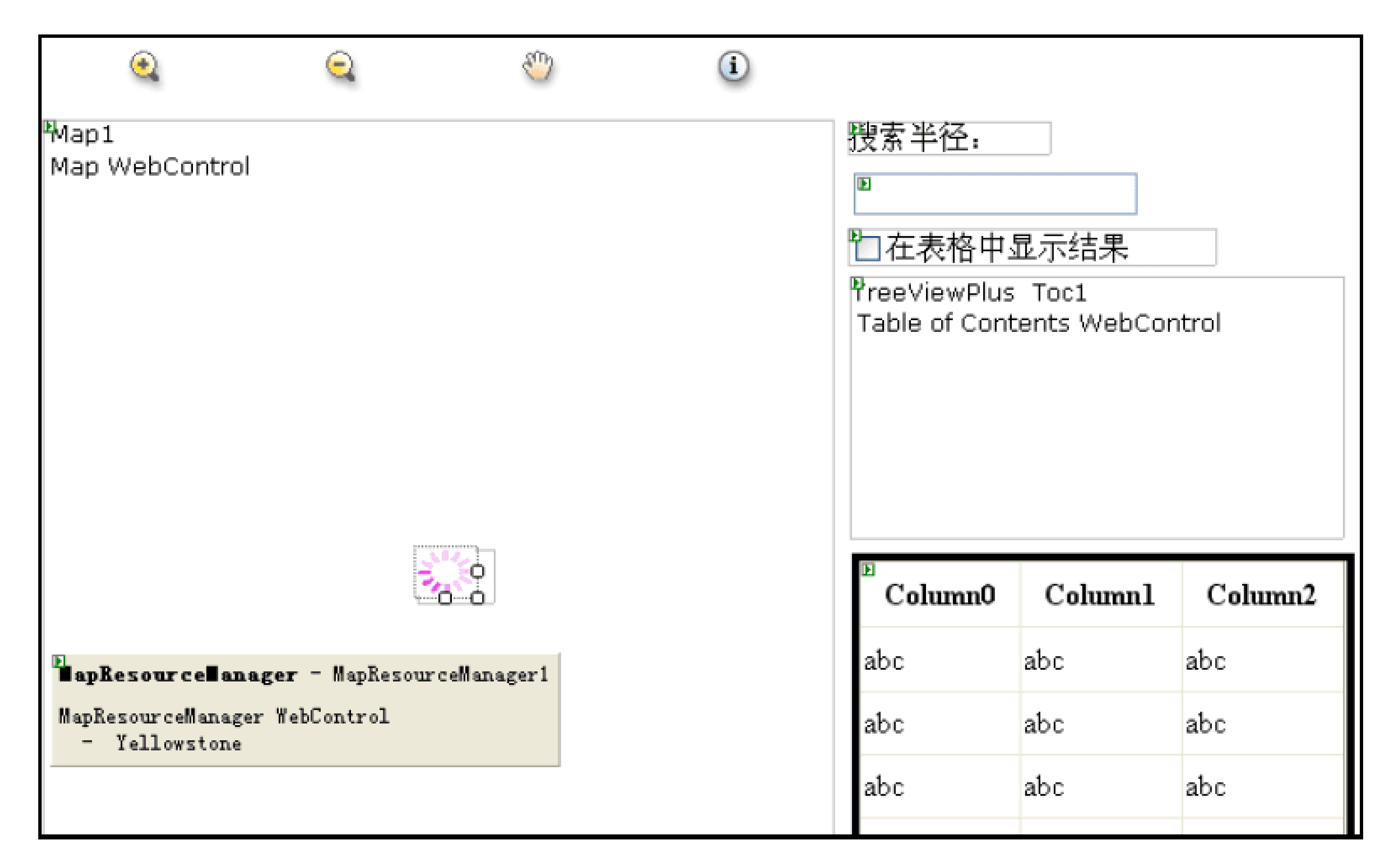


图 9.4　页面在设计期间的视图

自定义工具对应的类为 App_Code 目录下的 VegTool 类。在工程中增加该类。在类的头部加入如下命名空间的引用：

```
using ESRI.ArcGIS.ADF.Web.UI.WebControls;
using ESRI.ArcGIS.ADF.Web.UI.WebControls.Tools;
```

```
using ESRI.ArcGIS.ADF.Web.DataSources.ArcGISServer;
using ESRI.ArcGIS.ADF.Web.DataSources;
using ESRI.ArcGIS.Carto;
using ESRI.ArcGIS.esriSystem;
using ESRI.ArcGIS.Geodatabase;
using ESRI.ArcGIS.Server;
using ESRI.ArcGIS.Geometry;
using ESRI.ArcGIS.ADF.ArcGISServer;
```

将类的声明代码修改为如下代码，实现 IMapServerToolAction 接口：

```
public class VegTool : IMapServerToolAction
```

利用集成开发环境的代码自动生成功能生成 IMapServerToolAction 接口的代码框架，也就是 ServerAction 方法。在该方法中首先加入如下代码，得到服务器上下文对象与服务器对象：

```
ESRI.ArcGIS.ADF.Web.UI.WebControls.Map mapctrl = args.Control
    as ESRI.ArcGIS.ADF.Web.UI.WebControls.Map;

// 从地图控件中得到地图功能
MapFunctionality mapfunc =
    (MapFunctionality)mapctrl.GetFunctionality("Yellowstone");
MapResourceLocal mapres = (MapResourceLocal)mapfunc.MapResource;
IServerContext sc = mapres.ServerContextInfo.ServerContext;
IMapServer map = mapres.MapServer;
```

然后加入如下代码，从服务器对象中得到 Yellowstone 地图资源中的第一个图层对象，即植被图层：

```
IMapServerObjects mapobj = (IMapServerObjects)map;
IMap fgmap = mapobj.get_Map(map.DefaultMapName);
IFeatureLayer fl = (IFeatureLayer)fgmap.get Layer(0);
IFeatureClass fc = fl.FeatureClass;
```

接着加入如下代码，根据用户在地图上单击的点，得到 ArcObjects 的点对象

```
PointEventArgs pargs = (PointEventArgs)args;
ESRI.ArcGIS.ADF.Web.Geometry.Point inpt =
   ESRI.ArcGIS.ADF.Web.Geometry.Point.ToMapPoint(pargs.ScreenPoint,
                                          mapctrl.Extent,
                                          (int)mapctrl.Width.Value,
                                          (int)mapctrl.Height.Value);
IPoint pt = (IPoint)sc.CreateObject("esriGeometry.Point");
pt.X = inpt.X;
pt.Y = inpt.Y;
```

然后加入如下代码，得到用户在文本框中输入的半径，如果没有输入，则使用默认值：

```
string tbxvalue = (string)mapctrl.Page.Session["TextBox1Value"];
double distance = 0;
if (!Double.TryParse(tbxvalue, out distance)) {
    distance = 10000;
}
```

然后加入如下代码，在服务器上创建自定义的 COM 功能对象，然后调用其 sumVegetationType 方法，计算点缓冲区范围内不同植被的面积：

```
    VegCOM.IVegUtils vegutils = null;
    vegutils = (VegCOM.IVegUtils)sc.CreateObject("VegCOM.VegUtils");
    string fldName = "PRIMARY_";
    VegCOM.IVegResults vegresults = vegutils.sumVegetationType(ref fc, ref pt, ref
distance, ref fldName);
```

然后加入如下代码，绘制选择要素的几何图形：

```
    IGraphicElements comGraphics = vegresults.ResGraphics;
    GraphicElement[] proxyGraphics =
      (GraphicElement[])ESRI.ArcGIS.ADF.ArcGISServer.Converter.ComObjectToValue
Object(comGraphics, sc, typeof(GraphicElement[]));

    RgbColor rgb = new RgbColor();
    rgb.Red = 155;
    rgb.Green = 0;
    rgb.Blue = 0;
    rgb.AlphaValue = 255;

    SimpleLineSymbol sls = new SimpleLineSymbol();
    sls.Style = ESRI.ArcGIS.ADF.ArcGISServer.esriSimpleLineStyle.esriSLSSolid;
    sls.Color = rgb;
    sls.Width = 0.2;

    foreach (ESRI.ArcGIS.ADF.ArcGISServer.PolygonElement pe in proxyGraphics) {
       SimpleFillSymbol sfs = (SimpleFillSymbol)pe.Symbol;
       sfs.Outline = sls;
    }

    mapfunc.MapDescription.CustomGraphics = proxyGraphics;
```

最后加入显示结果的代码，如下所示：

```
    string cbxvalue = (string)mapctrl.Page.Session["CheckBox1Value"];
    if (bool.Parse(cbxvalue)) {
       IRecordSet rs = vegresults.Stats;
       ESRI.ArcGIS.ADF.ArcGISServer.RecordSet value_rs =
           ESRI.ArcGIS.ADF.Web.DataSources.ArcGISServer.Converter.ComObjectToV
alueObject(rs, sc, typeof(ESRI.ArcGIS.ADF.ArcGISServer.RecordSet))
           as ESRI.ArcGIS.ADF.ArcGISServer.RecordSet;
       System.Data.DataTable rsDatatable =
           ESRI.ArcGIS.ADF.Web.DataSources.ArcGISServer.Converter.ToDataTable(
value rs);
       System.Data.DataSet rsDataset = new System.Data.DataSet();
       rsDataset.Tables.Add(rsDatatable);
       mapctrl.Page.Session["VegDataset"] = rsDataset;
    }
    else {
       mapctrl.Page.Session["VegDataset"] = null;
```

```
    }

    mapctrl.Refresh();
```

切换到 Default.aspx.cs 文件中，首先在类的声明上加上 ICallbackEventHandler 接口，代码如下：

```
public partial class _Default : System.Web.UI.Page, ICallbackEventHandler
```

在类中加入两个字段：

```
public string callBackFunctionInvocation;
private string returnstring = "";
```

在类的 Page_Load 方法中，加入一些会话对象、控件的客户端函数，并获取一个对客户端函数的引用。代码如下：

```
protected void Page Load(object sender, EventArgs e) {
    if (!IsPostBack) {
        Session["TextBox1Value"] = "10000";
        Session["CheckBox1Value"] = "false";
        Session["VegDataset"] = null;
    }

    CheckBox1.Attributes.Add("onclick", "ChangeContext('CheckBox1')");
    TextBox1.Attributes.Add("onblur", "ChangeContext('TextBox1')");
    Map1.Attributes.Add("onmouseup", "ChangeContext('Map1')");
    callBackFunctionInvocation = ClientScript.GetCallbackEventReference(
        this, "message", "HandleResponse", "context", "postBackError", true);
}
```


回调相关处理方法的代码如下：

```
public void RaiseCallbackEvent(string eventArgs) {
    if (eventArgs.Contains("tbx")) {
        ChangeTextBoxServer(eventArgs);
    }
    else if (eventArgs.Contains("cbx")) {
        ChangeCheckBoxServer(eventArgs);
    }
    else if (eventArgs.Contains("map1")) {
        ChangeMapServer();
    }
}

public string GetCallbackResult() {
    return returnstring;
}

public void ChangeTextBoxServer(string ea) {
    char[] parser char = { ',' };
    string[] messages = ea.Split(parser char);
    string value = messages[1];
    Session["TextBox1Value"] = value;
}
```

```
public void ChangeCheckBoxServer(string ea) {
    char[] parser_char = { ',' };
    string[] messages = ea.Split(parser_char);
    string value = messages[1];
    Session["CheckBox1Value"] = value;
}

public void ChangeMapServer() {
    DataSet ds = (DataSet)Session["VegDataset"];

    if (ds != null) {
        GridView1.DataSource = ds.Tables[0];
        GridView1.DataBind();

        using (System.IO.StringWriter sw = new System.IO.StringWriter()) {
            HtmlTextWriter htw = new HtmlTextWriter(sw);
            GridView1.RenderControl(htw);
            htw.Flush();
            returnstring = sw.ToString();
        }
    }
}
```

切换到 Default.aspx 文件中，在<head>与</head>之间的代码段中加入如下一些 JavaScript 函数：

```
<script language="javascript" type="text/javascript">
     var context;

     function ChangeContext(controlname) {
        context = controlname;
        ChangeClient();
     }

     function ChangeClient() {
        var message;

        if (context == 'TextBox1') {
          var tbxvalue = document.getElementById('TextBox1').value;

          message = 'tbx';
          message += ',' + tbxvalue;
        }

        if (context == 'CheckBox1') {
            var cbxvalue = document.getElementById('CheckBox1').checked;

            message = 'cbx';
            message += ',' + cbxvalue;
        }

        if (context == 'Map1') {
            document.getElementById('loadingdiv').style.visibility =
```

```
    "visible";
            message = 'map1';
        }

        <%=callBackFunctionInvocation%>
    }

    function HandleResponse(returnvalue, ctx) {
        var griddiv = document.getElementById('griddiv');

        if (ctx == "Map1") {
            if (returnvalue != "") {
                griddiv.style.visibility = "visible";
                griddiv.innerHTML = returnvalue;
            }
            else if (returnvalue == "") {
                griddiv.style.visibility = "hidden";
            }
        }

        if (ctx == "Map1") {
            document.getElementById('loadingdiv').style.visibility="hidden";
        }
    }
</script>
```

编译并运行程序。选择“在表格中显示结果”复选框，然后先选择自定义工具，这时在地图上单击，服务器将计算以该点为圆心，指定半径范围内各种植被的面积。程序运行效果如图 9.5 所示。

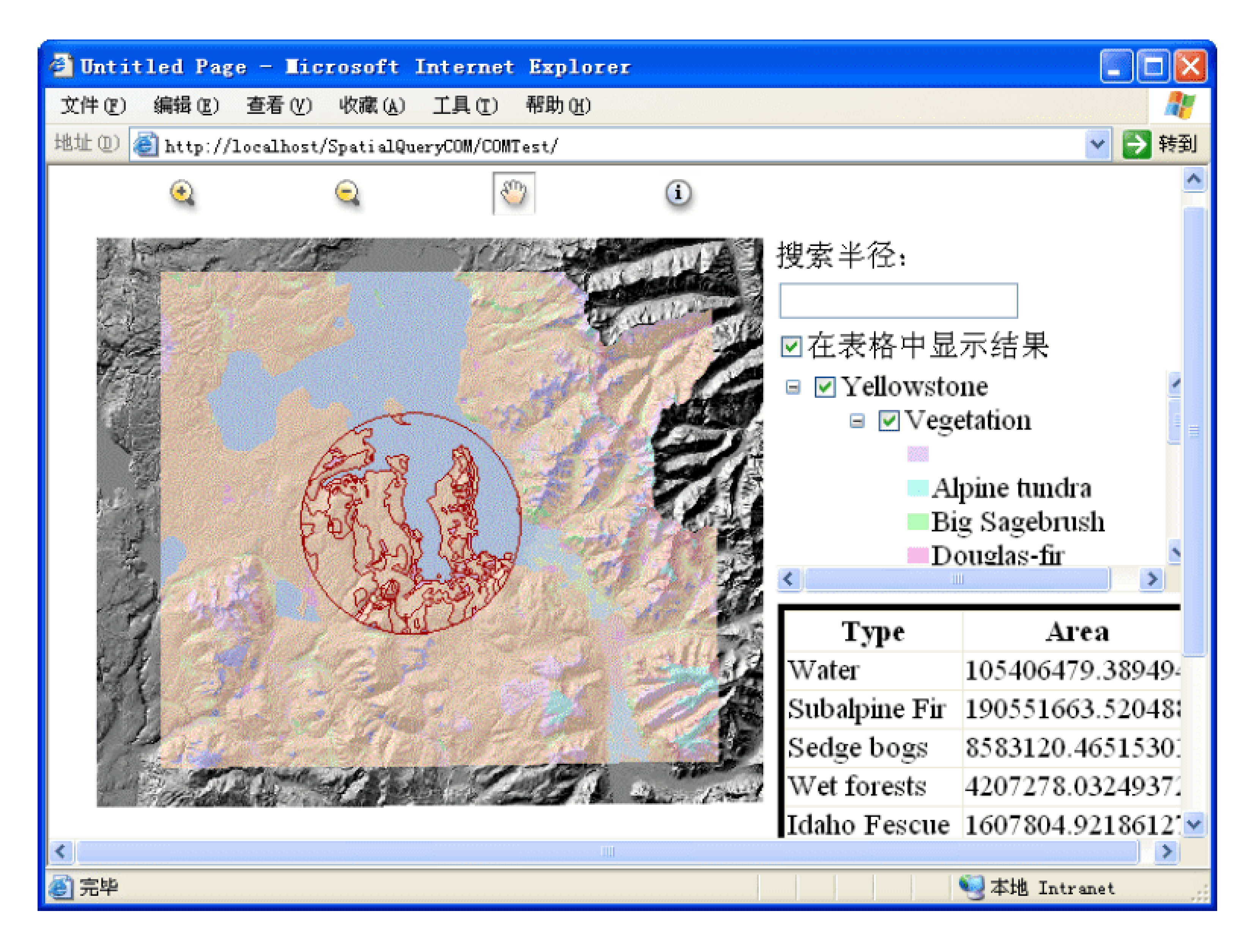


图 9.5 程序运行效果

9.3　服务器对象扩展

服务器对象扩展是扩展 GIS 服务器功能的另一种方法。服务器对象扩展的目的，就是将客户端的许多细粒度的调用，封装为粗粒度的方法，从而减少对 GIS 服务器的调用。

与 COM 对象相比，扩展服务器对象具有如下一些优势：

（1）开发人员不需要在上下文中显式创建服务器对象扩展。当服务器对象实例创建时，GIS 服务器同时创建服务器对象扩展。

（2）如果服务器对象扩展的功能需要消耗较多的资源，那么只有在服务器对象初始化时，才需要消耗该资源。

（3）可以使用缓存。

（4）ArcGIS Server 管理应用程序可以应用服务器对象扩展的配置对话框。

创建服务器对象扩展的流程如下：

（1）创建一个实现 IServerObjectExtension 接口与自定义接口的 COM 对象。还可选择实现其他一些接口，包括 IObjectConstruct、ILogSupport 与 IObjectActivate 接口。

（2）如果有必要，还可创建实现 IAGSSOEParameterPage 与 ICOMPropertyPage 接口的用户窗体，用于设置扩展的属性。

（3）在每台服务器对象容器计算机、服务器对象管理器计算机以及开发计算机上注册第一步创建的服务器扩展 COM 对象。

（4）使用 IServerObjectAdmin2 接口的 AddExtensionType 方法在 GIS 服务器上注册服务器对象扩展。

（5）如果必要，在每台 ArcGIS 桌面应用程序计算机上注册第二步创建的属性页。

（6）创建一个包含服务器对象扩展的服务器对象配置。

（7）创建使用服务器对象及其扩展的应用程序。

下面我们将通过一个实例演示如何创建服务器对象扩展。实现的功能与 9.2 节 COM 功能对象的功能类似。

9.3.1　创建服务器对象接口扩展

在 Visual Studio 2005 中，通过 File 菜单的 New Project 命令，创建一个 Class Library 的工程，命名为 VegSOEInterface。将解决方案命名为 VegSOE。

在工程中加入一个类，命名为 `VegSOEInterfaces`。删除其中所有的代码，然后在其中加入如下代码：

```
using System.Runtime.InteropServices;
using ESRI.ArcGIS.Geometry;
using ESRI.ArcGIS.Carto;
using ESRI.ArcGIS.Geodatabase;
```

```
namespace VegSOEInterfaces {
    [GuidAttribute("70a7fbc8-ab8e-4e3c-810f-4e0f46f62e49")]
    public interface IVegUtilsSOE {
        IVegResultsSOE sumVegetationType(IPoint pPoint, double dDistance);
    }

    [GuidAttribute("3c09de7a-c0ce-4c6f-b1b6-8100e712c1b2")]
    public interface IVegResultsSOE {
        IGraphicElements ResGraphics {
            get;
            set;
        }
        IRecordSet Stats {
            get;
            set;
        }
    }
}
```

上面的代码，定义了两个接口。其中 IVegUtilsSOE 接口定义了将要被 VegUtilsSOE 类实现的基本框架。IVegResultsSOE 接口定义了将要被 VegResultsSOE 类实现的基本框架。由于两个接口都要注册为 COM 对象，因此需要指定一个唯一的 GUID。

在工程中加入 ESRI.ArcGIS.Carto、ESRI.ArcGIS.Display、ESRI.ArcGIS.Geodatabase、ESRI.ArcGIS.Geometry、ESRI.ArcGIS.Server、ESRI.ArcGIS.System 与 System.EnterpriseServices 程序集的引用。

打开该工程的 AssemblyInfo.cs 文件，将 ComVisibleAttribute 属性的值由原来的 false 修改为 true。即将：

```
[assembly: ComVisible(false)]
```

修改为：

```
[assembly: ComVisible(true)]
```

9.3.2 实现服务器对象扩展

在当前解决方案中加入名为 VegSOE 的 Class Library 类型的工程。在工程中加入 ESRI.ArcGIS.Carto、ESRI.ArcGIS.Catalog、ESRI.ArcGIS.CatalogUI、ESRI.ArcGIS.Display、ESRI.ArcGIS.Framework、ESRI.ArcGIS.Geodatabase、ESRI.ArcGIS.Geometry、ESRI.ArcGIS.Server、ESRI.ArcGIS.System 与 System.EnterpriseServices 程序集的引用。还需要加入 VegSOEInterface 工程的引用。

将默认加入的 Class1.cs 文件更名为 VegResultsSOE.cs，集成开发环境将自动将类名改为 VegResultsSOE。在类的头部加入如下一些命名空间的引用：

```
using System.Collections.Specialized;
using System.Runtime.InteropServices;
```

```
using System.EnterpriseServices;
using ESRI.ArcGIS.Carto;
using ESRI.ArcGIS.Geodatabase;
```

VegResultsSOE 类的实现代码很简单，代码如下：

```
[AutomationProxy(true)]
[ClassInterface(ClassInterfaceType.None)]
[GuidAttribute("05922ca8-1a80-4502-8bbf-b3c0637b80ac")]
public class VegResultsSOE :
ServicedComponent, VegSOEInterfaces.IVegResultsSOE
{
    private IGraphicElements m resGraphics;
    private IRecordSet m resStats;

    public IGraphicElements ResGraphics {
        get {
            return m_resGraphics;
        }
        set {
            m resGraphics = (IGraphicElements)value;
        }
    }

    public IRecordSet Stats {
        get {
            return m_resStats;
        }
        set {
            m resStats = (IRecordSet)value;
        }
    }
}
```

在工程中新加入名为 VegUtilsSOE 的类。在类的头部加入如下命名空间的引用：

```
using System.Runtime.InteropServices;
using System.EnterpriseServices;
using ESRI.ArcGIS.Carto;
using ESRI.ArcGIS.Display;
using ESRI.ArcGIS.esriSystem;
using ESRI.ArcGIS.Geodatabase;
using ESRI.ArcGIS.Geometry;
using ESRI.ArcGIS.Server;
```

修改类的声明代码：

```
[AutomationProxy(true)]
[ClassInterface(ClassInterfaceType.None)]
[GuidAttribute("87176523-1ede-4fe6-abe0-66481ac7d04b")]
public class VegUtilsSOE : ServicedComponent, VegSOEInterfaces.IVegUtilsSOE,
    IServerObjectExtension, IObjectConstruct, ILogSupport, IObjectActivate
```

该类继承 ServicedComponent 类，还需要实现 VegSOEInterfaces.IVegUtilsSOE、IServerObjectExtension、IObjectConstruct、ILogSupport 与 IObjectActivate 接口。

服务器对象扩展初始化时各接口中方法的调用顺序如图 9.6 所示。

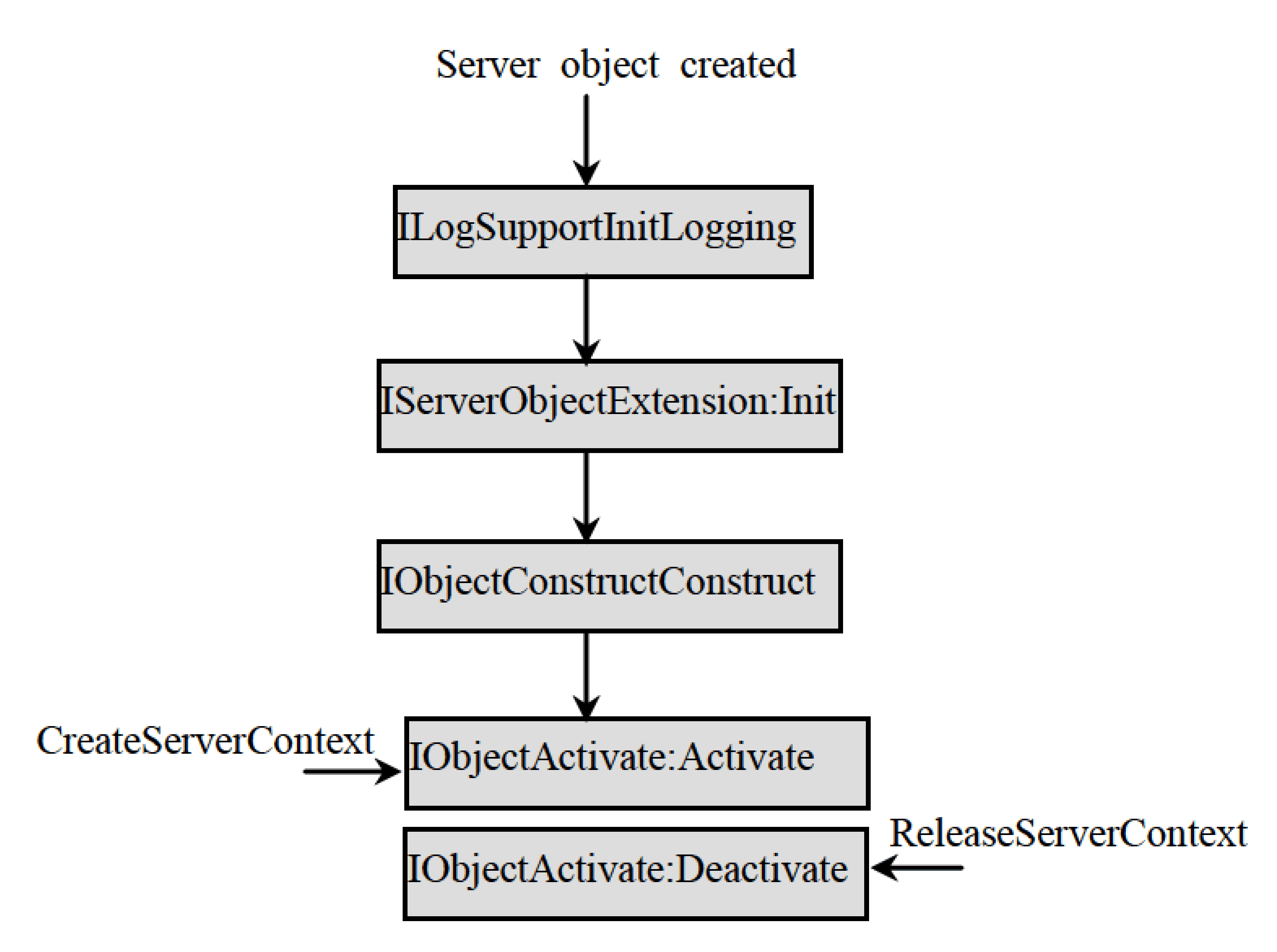


图 9.6 服务器对象扩展初始化时各方法的调用顺序

ServicedComponent 是使用 COM 服务的所有类的基类。只需要进行声明，而不需要任何额外的代码。

如果希望服务器对象扩展能在 GIS 服务器日志文件中记录日志消息，则必须实现 ILogSupport 接口。该接口中只有一个 InitLogging 方法。当服务器对象扩展创建时，调用该方法，获取 GIS 服务器日志对象的引用。一旦有了服务器日期的引用，便可随时调用 AddMessage 方法往日志中加入消息。在 VegUtilsSOE 中加入一服务器日志引用字段以及 InitLogging 方法的实现，代码如下：

```
private ILog m log;
public void InitLogging(ILog log) {
    m log = log;
}
```

服务器对象扩展中最重要的接口是 IServerObjectExtension。所有服务器对象扩展必须实现该接口。该接口中包含两个方法，分别是 Init 与 Shutdown。从这两个方法的名称，可以看出，该接口用于服务器对象管理扩展的整个生命周期。服务器对象扩展必须通过服务器对象帮助器（IServerObjectHelper）才能调用服务器对象的任何方法（当服务器对象扩展创建时调用 Init 方法。Shutdown 方法通过服务器对象的上下文被关闭，并即将销毁，因此需要在该方法中释放服务器对象帮助器的引用）。在 VegUtilsSOE 类中加入服务器对象帮助器的引用及上述两个方法的实现，代码如下：

```
private IServerObjectHelper m SOH;

public void InitLogging(ILog log) {
    m_log = log;
```

```
    }

    public void Init(IServerObjectHelper pSOH) {
        m_SOH = pSOH;
        m_log.AddMessage(3, 8000, "VegUtilsSOE custom message. Init called");
    }

    public void Shutdown() {
        m_log.AddMessage(3, 8000, "VegUtilsSOE custom message. Shutdown called");
        m_SOH = null;
        m_layer = null;
        m_log = null;
    }
```

如果服务器对象扩展包含配置属性或需要其他额外的初始化处理，则需要实现接口 IObjectConstruct。该接口仅包含一个方法 Construct。服务器对象扩展创建时，在调用 IServerObjectExtension 接口的 Init 方法之后，调用 Construct 方法。

配置属性存储在服务器对象的配置文件中。配置文件的命名方式为“服务名称.服务器对象类型.cfg”，存储在 SOM 计算机的 ArcGIS 安装目录的 Server\user\cfg 路径中。例如，名为 Yellowstone 的地图服务的配置文件的名称为 Yellowstone.MapServer.cfg。在我们这个实例中，需要从配置文件中读入图层与字段属性。在 VegUtilsSOE 类中加入如下代码：

```
    private ILayer m_layer;
    private string m_layerName;
    private string m_fieldName;

    public void Construct(IPropertySet props) {
        try {
            m_layerName = props.GetProperty("LayerName") as string;
            m_fieldName = props.GetProperty("FieldName") as string;
        }
        catch (Exception ex) {
            m_log.AddMessage(1, 8000,
                "VegUtilsSOE custom error. Error reading properties: "
                + ex.Message + " " + props.Count.ToString());
            return;
        }

        try {
            IMapServer ms = (IMapServer)m_SOH.ServerObject;
            IMapServerObjects mso = (IMapServerObjects)ms;
            IMap map = mso.get_Map(ms.DefaultMapName);
            UID ltype = new UIDClass();
            ltype.Value = "{E156D7E5-22AF-11D3-9F99-00C04F6BC78E}";
            IEnumLayer el = map.get_Layers(ltype, true);
            el.Reset();

            ILayer l = null;
            while ((l = el.Next()) != null) {
```

```
                if (l.Name == m_layerName) {
                    m_layer = l;
                    break;
                }
            }

            if (m layer == null) {
                m_log.AddMessage(1, 8000, "VegUtilsSOE custom error: Layer "
                    + m_layerName + " not found.");
                return;
            }

            IFeatureLayer fl = (IFeatureLayer)m layer;
            IFeatureClass fc = fl.FeatureClass;

            if (fc.FindField(m_fieldName) == -1)
                m_log.AddMessage(1, 8000, "VegUtilsSOE custom error: Field "
                    + m_fieldName + " not found in layer " + m_layerName);
            else
                m log.AddMessage(3, 8000, "VegUtilsSOE successfully initialized.");

        }
        catch (Exception ex) {
            m_log.AddMessage(1, 8000,
                "VegUtilsSOE custom error: Failed to initialize extension: "
                + ex.Message + "::" + ex.StackTrace.Length.ToString());
        }
    }
```


IServerObjectExtension 接口的 Init 方法与 IObjectConstruct 接口的 Construct 方法，只有在服务器对象扩展创建时才被调用。然而如果需要在每次服务器上下文获取或释放时（即 Web 端调用 CreateServerContext 与 ReleaseContext），运行服务器对象扩展，则需要实现接口 IObjectActivate。该接口包含两个方法，分别是 Activate 与 Deactivate。当 Web 端每次调用 CreateServerContext 方法时，调用 Activate 方法，而每次调用 ReleaseContext 释放服务器上下文时调用 Deactivate 方法。在 VegUtilsSOE 类中加入实现 IServerObjectExtension 接口方法的代码：

```
    public void Deactivate() {
        m log.AddMessage(3, 8000,
            "VegUtilsSOE custom message. Deactivate called");
    }

    public void Activate() {
        m_log.AddMessage(3, 8000,
            "VegUtilsSOE custom message. Activate called");
    }
```

下面加入 IVegUtilsSOE 接口需要实现的方法，实现统计指定字段中各类地物的面积。代码如下：

```
    public VegSOEInterfaces.IVegResultsSOE sumVegetationType(
```

```
    IPoint pPoint, double dDistance)
    {
        if (m layer == null) {
            m log.AddMessage(1, 8000, "VegUtilsSOE custom error: layer not found");
            return null;
        }

        IFeatureLayer fl = (IFeatureLayer) m_layer;
        IFeatureClass pVegClass = fl.FeatureClass;

        ITopologicalOperator pTopoOp = (ITopologicalOperator) pPoint;
        IGeometry pGeom = pTopoOp.Buffer(dDistance);

        ISpatialFilter pSFilter = new SpatialFilter();
        pSFilter.Geometry = pGeom;
        pSFilter.SpatialRel = esriSpatialRelEnum.esriSpatialRelIntersects;
        pSFilter.GeometryField = pVegClass.ShapeFieldName;

        IFeatureCursor pFCursor = pVegClass.Search(pSFilter, true);

        pTopoOp = (ITopologicalOperator) pGeom;
        int lPrim = pVegClass.FindField(m fieldName);

        ListDictionary dict = new ListDictionary();

        ISimpleFillSymbol pSFS = newFillS();
        IGraphicElements pGraphics = new GraphicElements();

        IFeature pFeature;
        while ((pFeature = pFCursor.NextFeature()) != null) {
            IFillShapeElement pFE = (IFillShapeElement) new PolygonElement();
            IElement pElement = pFE as IElement;

            IGeometry pNewGeom = pTopoOp.Intersect(pFeature.Shape,
                                 esriGeometryDimension.esriGeometry2Dimension);
            pElement.Geometry = pNewGeom;
            pFE.Symbol = pSFS;
            IGraphicElement ge = (IGraphicElement) pFE;
            pGraphics.Add(ge);

            IArea pArea = pNewGeom as IArea;
            string sType = pFeature.get Value(lPrim) as string;
            if (dict.Contains(sType))
                dict[sType] = (double)dict[sType] + pArea.Area;
            else
                dict[sType] = pArea.Area;
        }

        IRecordSet psumRS = sumRS(dict);
```

```
        VegSOEInterfaces.IVegResultsSOE pRes = new VegResultsSOE();
        pRes.ResGraphics = pGraphics;
        pRes.Stats = psumRS;

        return pRes;
    }

    private IRecordSet sumRS(
    System.Collections.Specialized.ListDictionary dict)
    {
        IRecordSet pNewRs = new RecordSet();
        IRecordSetInit prsInit = pNewRs as IRecordSetInit;

        IFields pFields = new Fields();
        IFieldsEdit pFieldsEdit = pFields as IFieldsEdit;
        pFieldsEdit.FieldCount_2 = 2;

        IField pField = new Field();
        IFieldEdit pFieldEdit = pField as IFieldEdit;
        pFieldEdit.Name_2 = "Type";
        pFieldEdit.Type_2 = esriFieldType.esriFieldTypeString;
        pFieldEdit.Length_2 = 50;
        pFieldsEdit.set_Field(0, pField);

        pField = new Field();
        pFieldEdit = pField as IFieldEdit;

        pFieldEdit.Name_2 = "Area";
        pFieldEdit.Type_2 = esriFieldType.esriFieldTypeDouble;
        pFieldsEdit.set_Field(1, pField);

        prsInit.CreateTable(pFields);

        ICursor pIC = prsInit.Insert();
        IRowBuffer pRowBuf = prsInit.CreateRowBuffer();

        System.Collections.IDictionaryEnumerator myEnumerator =
            dict.GetEnumerator();
        while (myEnumerator.MoveNext()) {
            pRowBuf.set_Value(0, myEnumerator.Key);
            pRowBuf.set_Value(1, myEnumerator.Value);
            pIC.InsertRow(pRowBuf);
        }

        return pNewRs;
    }

    private ISimpleFillSymbol newFillS() {
        ISimpleLineSymbol pSLS = new SimpleLineSymbol();
        IRgbColor pcolor = new RgbColor();
```

```
        pcolor.Red = 0;
        pcolor.Green = 255;
        pcolor.Blue = 0;
        pSLS.Color = pcolor;
        pSLS.Style = esriSimpleLineStyle.esriSLSSolid;
        pSLS.Width = 2;

        ISimpleFillSymbol pSFS = new SimpleFillSymbol();
        pSFS.Outline = pSLS;
        pSFS.Style = esriSimpleFillStyle.esriSFSHollow;

        return pSFS;
    }
```

打开该工程的AssemblyInfo.cs文件，将ComVisibleAttribute属性的值由原来的false修改为true。即将：

```
    [assembly: ComVisible(false)]
```

修改为：

```
    [assembly: ComVisible(true)]
```

9.3.3　创建服务器对象扩展的属性页

在解决方案中增加一名为 VegSOEProps 的 Class Library 类型的工程。在工程中加入 ESRI.ArcGIS.Carto、ESRI.ArcGIS.System、ESRI.ArcGIS.Geodatabase、ESRI.ArcGIS.Geometry、ESRI.ArcGIS.Catalog、ESRI.ArcGIS.CatalogUI、ESRI.ArcGIS.Framework 与 ESRI.ArcGIS.Server 程序集的引用。

在当前工程中加入一名为 FormVegProps 的 Windows Form。在该窗体中增加三个标签控件与两个下拉列表框控件。在窗体设计期间的视图如图 9.7 所示。

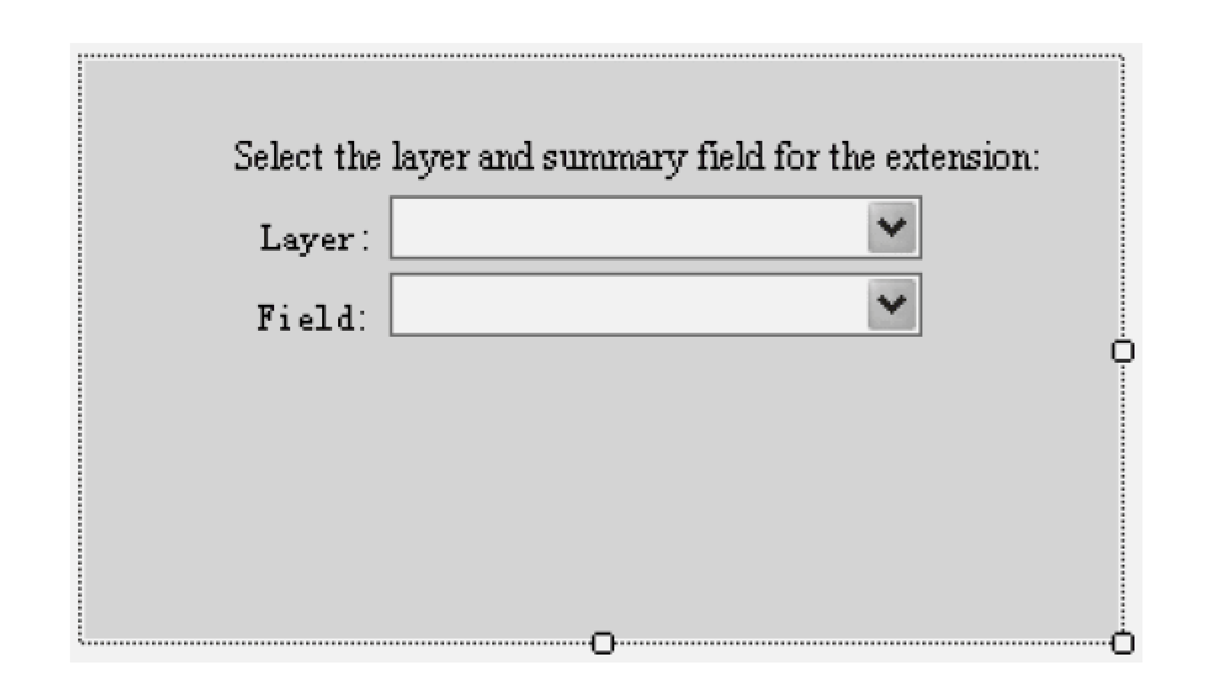


图 9.7　服务器对象扩展的属性页在设计期间的视图

打开该窗体的代码，在头部加入如下命名空间的引用：

```
    using ESRI.ArcGIS.Carto;
    using ESRI.ArcGIS.esriSystem;
```

```
using ESRI.ArcGIS.Geodatabase;
using ESRI.ArcGIS.Geometry;
```

增加返回窗体句柄的方法，在 ArcCatalog 中显示属性页时需要该方法。代码如下：

```
public int getHWnd() {
    return this.Handle.ToInt32();
}
```

窗体需要引用地图文档、图层以及字段名称。因此加入如下一些字段及对应属性的代码：

```
private IMapDocument m_map;
private string m_layer;
private string m field;

public string theLayer {
    get {
        return m layer;
    }
    set {
        ComboLayers.Text = value;
        m layer = ComboLayers.Text;
    }
}

public string theField {
    get {
        return m_field;
    }
    set {
        ComboFields.Text = value;
        m field = ComboFields.Text;
    }
}

public void setMap(string sName) {
    ComboLayers.Items.Clear();
    ComboFields.Items.Clear();

    m map = new MapDocumentClass();
    m map.Open(sName, null);

    IMap map = m_map.get_Map(0);

    UID id = new UIDClass();
    id.Value = "{E156D7E5-22AF-11D3-9F99-00C04F6BC78E}";
    IEnumLayer el = map.get Layers(id, true);

    ILayer l = null;
    while ((l = el.Next()) != null) {
        if (l is IFeatureLayer) {
            IFeatureLayer fl = (IFeatureLayer) l;
```

```
            IFeatureClass fc = fl.FeatureClass;

            if (fc.ShapeType == esriGeometryType.esriGeometryPolygon
               && fc.FeatureType == esriFeatureType.esriFTSimple)
               ComboLayers.Items.Add(l.Name);
         }
      }

      if (m_layer == null)
         ComboLayers.SelectedIndex = 0;
      else {
         IEnumerator itemenum = ComboLayers.Items.GetEnumerator();
         while (itemenum.MoveNext()) {
            string lname = itemenum.Current.ToString();
            if (lname == m layer)
               ComboLayers.SelectedIndex =
                  ComboLayers.Items.IndexOf(itemenum.Current);
         }
      }

   }
```

加入处理下拉列表框改变选择的事件响应方法的代码：

```
   private void ComboLayers_SelectedIndexChanged(object sender, EventArgs e)
   {
      ComboFields.Items.Clear();

      IMap map = m map.get Map(0);

      UID id = new UIDClass();
      id.Value = "{E156D7E5-22AF-11D3-9F99-00C04F6BC78E}";
      IEnumLayer el = map.get_Layers(id, true);

      ILayer l = null;
      while ((l = el.Next()) != null) {
         if (l is IFeatureLayer && l.Name == ComboLayers.Text) {
            IFeatureLayer fl = (IFeatureLayer) l;
            IFields flds = fl.FeatureClass.Fields;

            for (int i = 0; i < flds.FieldCount; i++) {
               IField fld = flds.get_Field(i);
               ComboFields.Items.Add(fld.Name);
            }
         }
      }

      if (m field == null)
         ComboFields.SelectedIndex = 0;
      else {
         IEnumerator itemenum = ComboFields.Items.GetEnumerator();
```

```
            while (itemenum.MoveNext()) {
                string fname = itemenum.Current.ToString();
                if (fname == m_field)
                    ComboFields.SelectedIndex =
                       ComboFields.Items.IndexOf(itemenum.Current);
            }
        }

        m_layer = ComboLayers.SelectedItem.ToString();

    }

    private void ComboFields_SelectedIndexChanged(object sender, EventArgs e){
        m_field = ComboFields.Text;
    }

    private void FormVegProps_FormClosed(object sender, FormClosedEventArgs e){
        m_map.Close();
        m_map = null;
    }
```

窗体只是用于呈现界面，真正实现设置还需要一个类。在工程中新增加名为 VegSOEProps 的类。在该类的头部加入如下命名空间的引用：

```
    using System.Runtime.InteropServices;
    using ESRI.ArcGIS.Carto;
    using ESRI.ArcGIS.Catalog;
    using ESRI.ArcGIS.CatalogUI;
    using ESRI.ArcGIS.esriSystem;
    using ESRI.ArcGIS.Server;
    using ESRI.ArcGIS.Framework;
    using ESRI.ArcGIS.Utility.CATIDs;
```

将类的声明修改为如下代码：

```
    [GuidAttribute("EB085B89-D01F-448c-98F0-448F0BECB0BF")]
    public class VegSOEProps : IComPropertyPage, IAGSSOEParameterPage
```

在类中增加如下几个字段：

```
    private IPropertySet m_props;
    private IPropertySet m_soprops;
    private string m_exttype;
    private string m_sotype;
    private FormVegProps propertyPage;
```

当在 ArcCatalog 中需要创建与显示属性页时，需要调用该属性页类。VegSOEProps 类的构造函数用于初始化成员变量。在该函数中，创建一个 FormVegProps 窗体实例，并赋予 propertyPage 字段。服务器对象扩展将被注册用于 MapServer 类型的服务器对象，也就是 m_sotype 字段指定的值，而扩展对象的名称，即 VegUtilitiesSOE，使用 m_exttype 字段存储，该值必须与在 ArcGIS Server 中注册时使用的名称一致。析构函数用于显式销毁属性页。构造函数与析构函数的代码如下：

```
public VegSOEProps() {
    propertyPage = new FormVegProps();
    m sotype = "MapServer";
    // 必须与注册 SOE 的名称一致
    m exttype = "VegUtilitiesSOE";
}

~VegSOEProps() {
    propertyPage.Dispose();
    propertyPage = null;
}
```

属性页类需要实现 IComPropertyPage 与 IAGSSOEParameterPage 两个接口。

IComPropertyPage 接口定义了 ArcCatalog 与属性页交互的框架。在类中加入如下代码，实现该接口：

```
public bool IsPageDirty {
    get {
        return false;
    }
}

public void Cancel() {
}

public int get_HelpContextID(int controlID) {
    return 0;
}

public string Title {
    get {
        return null;
    }
    set {
    }
}

public void SetObjects(ISet objects) {
}

public int Width {
    get {
        return 0;
    }
}

public int Priority {
    get {
        return 0;
    }
```

```
        set {
        }
    }

    public void Apply() {
    }

    public IComPropertyPageSite PageSite {
        set {
        }
    }

    public void Deactivate() {
    }

    public int Height {
        get {
            return 0;
        }
    }

    public void Show() {
        propertyPage.Show();
    }

    public string HelpFile {
        get {
            return null;
        }
    }

    public int Activate() {
        return propertyPage.getHWnd();
    }

    public bool Applies(ISet objects) {
        return false;
    }

    public void Hide() {
        propertyPage.Hide();
    }
```

IAGSSOEParameterPage 接口中定义了一组属性，自定义的服务器对象扩展需要显式实现。该接口通过在该类中定义的服务器对象及其扩展属性，以及地图服务器对象的配置文件，更新属性页与配置文件。服务器对象的配置属性存储在服务器对象配置文件中。而服务器对象扩展的属性也存储在该配置文件中。在本实例中，需要将我们自定义的服务器对象扩展的属性，即 LayerName 与 FieldName 写到配置文件中，此外需要从该配置文件中读入三个属性，分别是自定义服务器对象扩展的两个属性与一个服务器对象自身的属性（FilePath）。

ServerObjectProperties 属性只存储服务器对象的属性，而不包括扩展的属性。如果服务器对象扩展已经被配置，那么 LayerName 与 FieldName 属性中就有值。如果没有，则只有服务器对象的地图文档的路径可使用。当属性页窗体打开时，调用 ServerObjectProperties 属性的 set 方法。

ExtensionProperties 属性管理服务器对象扩展文件中属性的访问。在本实例中，如果服务器对象扩展可使用，那么在服务器对象配置文件中会加入如下属性：

```
<Extension>
    <TypeName>VegUtilitiesSOE </TypeName>
    <Enabled>true</Enabled>
    <Properties>
        <LayerName>Vegetation</LayerName>
        <FieldName>PRIMARY </FieldName>
    </Properties>
    <Info>
        <WebEnabled>true</WebEnabled>
        <WebCapabilities></WebCapabilities>
    </Info>
</Extension>
```

可以通过属性页改变 LayerName 与 FieldName 属性的值。当在 ArcCatalog 中选择了某服务器对象扩展，那么这些属性就会被加入到地图服务器对象的配置文件中。通过 ExtensionProperties 属性的 get 方法在配置文件中增加属性。如果属性值已经加入到配置文件中，那么该属性的 set 方法就会读入属性值。

当属性页初始化时，设置 ServerObjectExtensionType 与 ServerObjectType 属性。

```
public IPropertySet ServerObjectProperties {
    get { return m_soprops; }
    set {
        m soprops = value;

        try {
          propertyPage.theLayer =
                m props.GetProperty("LayerName").ToString();
          propertyPage.theField =
                m_props.GetProperty("FieldName").ToString();
          propertyPage.setMap(m soprops.GetProperty("FilePath").ToString());
        }
        catch {
          propertyPage.setMap(m soprops.GetProperty("FilePath").ToString());
        }
    }
}

public IPropertySet ExtensionProperties {
    get {
        m props.SetProperty("LayerName", propertyPage.theLayer);
        m props.SetProperty("FieldName", propertyPage.theField);
        return m props;
    }
```

```
        set {
            m_props = value;
        }
    }

    public string ServerObjectExtensionType {
        get {
            return m_exttype;
        }
    }

    public string ServerObjectType {
        get {
            return m_sotype;
        }
    }
```

为了能在 ArcCatalog 中访问属性页，需要将其注册为 AGS Extension Parameter Pages 组件类型。当属性页类注册为 COM 时，实现该注册。在类中加入如下代码进行注册：

```
    [ComRegisterFunction()]
    static void RegisterFunction(String regKey) {
        Microsoft.Win32.Registry.ClassesRoot.CreateSubKey(regKey.Substring(18)
            + "\\Implemented Categories\\"
            + "{A585A585-B58B-4560-80E3-87A411859379}");
    }

    [ComUnregisterFunction()]
    static void UnregisterFunction(String regKey) {
        Microsoft.Win32.Registry.ClassesRoot.CreateSubKey(regKey.Substring(18)
            + "\\Implemented Categories\\"
            + "{A585A585-B58B-4560-80E3-87A411859379}");
    }
```

要查看注册的 ArcGIS 组件类型，可以运行 ArcGIS 安装目录下的 bin 路径中的 categoris.exe 程序，并选择某类型。该程序运行界面如图 9.8 所示。

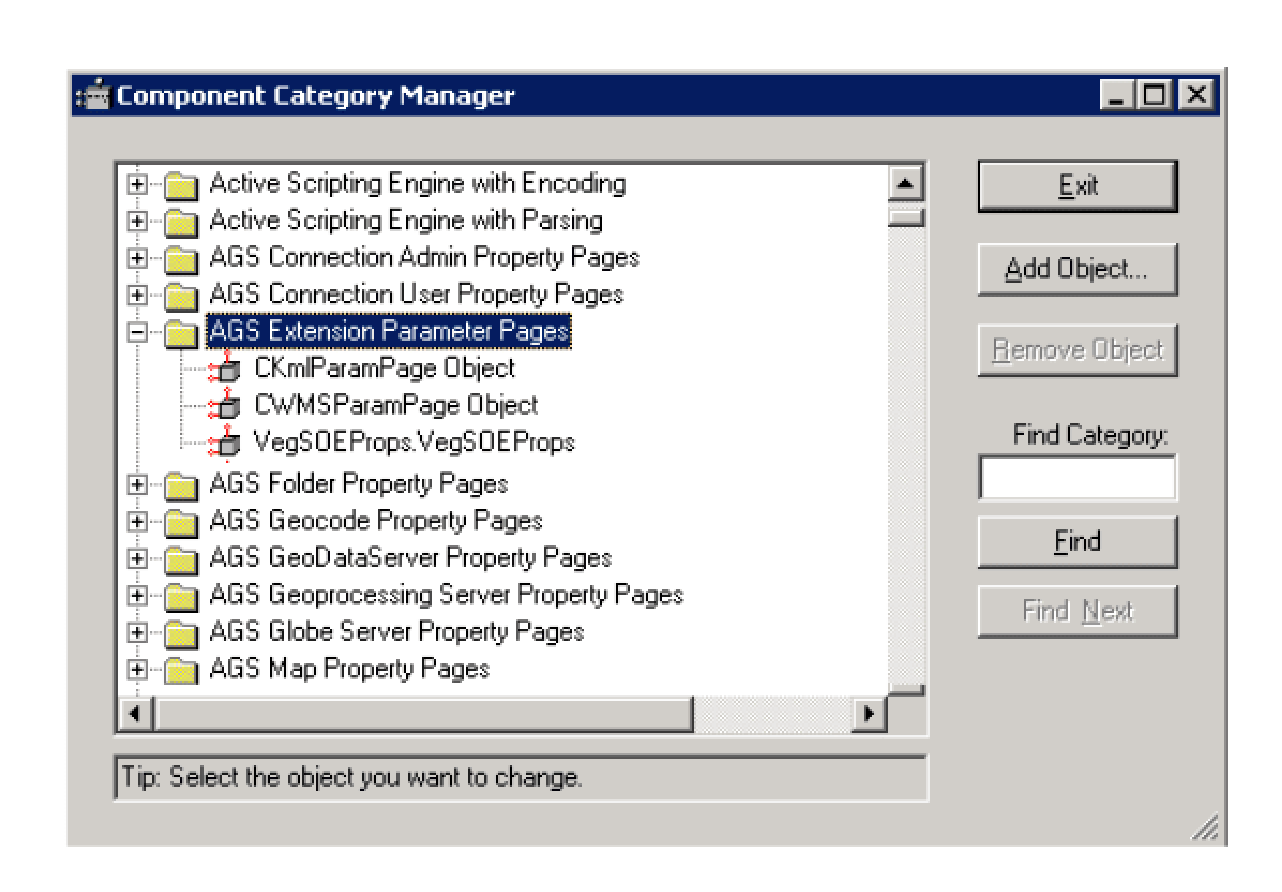


图 9.8 ArcGIS 组件类型管理器

打开该工程的 AssemblyInfo.cs 文件，将 ComVisibleAttribute 属性的值由原来的 false 修改为 true。即将：

```
[assembly: ComVisible(false)]
```

修改为：

```
[assembly: ComVisible(true)]
```

编译工程。

9.3.4　注册自定义服务器对象扩展

自定义服务器对象扩展组件必须进行两类注册。一类是程序集必须在使用计算机上使用.NET\COM 注册。另一类是必须在 ArcGIS 服务器上注册服务器对象扩展。

通过前面的代码，创建了三个.NET 程序集。

- ❑ VegSOECSharp.dll——包含了服务器对象扩展类及业务逻辑；
- ❑ VegSOEInterfacesCSharp.dll——包含了服务器对象扩展实现的接口；
- ❑ VegSOEPropsCSharp.dll——包含配置服务器对象扩展的属性页。

图 9.9 阐述了哪些应用程序或进程访问哪些组件。要注意的是，SOM 运行的计算机不需要访问本实例创建的自定义组件，但是需要在该计算机上注册自定义服务器对象扩展。

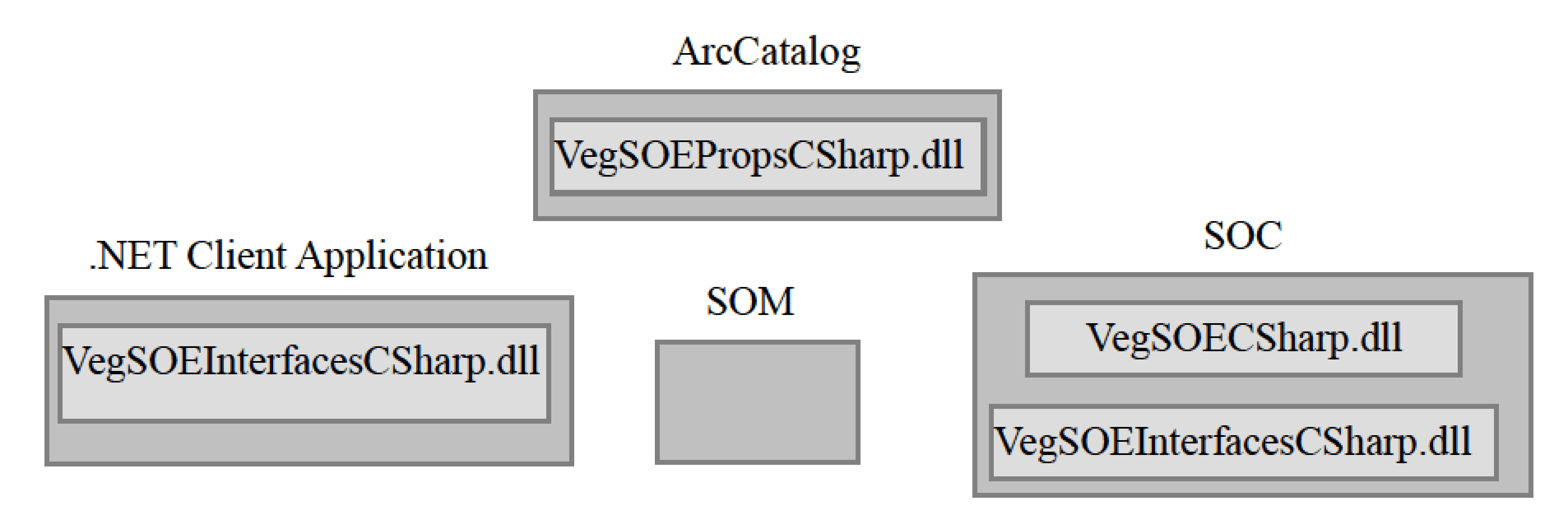


图 9.9　各组件应该分布的计算机

VegSOE 程序需要在所有服务器对象容器计算机（ArcSOC.exe）上注册。ArcSOC.exe 是 COM 客户端，可以访问其他 COM 对象与类型。打开 Visual Studio 2005 命令行工具，切换到 VegSOECSharp.dll 所在的目录，使用如下命令注册：

```
regasm VegSOE.dll /codebase
```

VegSOEInterfaces 程序集定义了服务器对象扩展的接口，因此也需要在所有服务器对象容器计算机上注册。注册命令如下：

```
regasm VegSOEInterfaces.dll /tlb:VegSOEInterfacesCSharp.tlb
```

regasm 工具从程序集中读取元数据。由于程序集只包含接口类型，因此必须使用/tlb 选项生成一类型库。

VegSOEProps 程序集必须在 ArcCatalog 运行的计算机上注册。注册命令如下：

```
regasm VegSOEPropsCSharp.dll /codebase
```

9.3.5 在 GIS 服务器上注册服务器对象扩展

服务器对象扩展需要在 ArcGIS 服务器上注册，特别是需要在服务器对象管理器中注册。需要使用 ESRI.ArcGIS.Server 程序集中的 ArcObjects 组件，将服务器对象扩展可访问性与属性等信息，加入到 ArcGIS 服务器的配置文件中。

在解决方案中创建一个名为 RegisterSOE 的控制台类型的工程。并在工程中加入 ESRI.ArcGIS.ADF.Connection 与 ESRI.ArcGIS.Server 程序集的引用。

打开 Program.cs 文件，在其中加入如下命名空间的引用：

```
using ESRI.ArcGIS.ADF.Connection.AGS;
using ESRI.ArcGIS.Server;
```

在 Main 方法中加入如下代码：

```
AGSServerConnection gisconnection = new AGSServerConnection();
gisconnection.Host = "localhost";
gisconnection.Connect();
IServerObjectAdmin2 soa =
   (IServerObjectAdmin2)gisconnection.ServerObjectAdmin;
IServerObjectExtensionType soet = soa.CreateExtensionType();
```

上述代码创建了一个 AGSServerConnection 类的实例，用于初始化与 SOM 的连接。然后指定 SOM 运行计算机的名称或 IP 地址，建立连接，并利用 IServerObjectAdmin2 接口创建一个扩展类型。

需要设置新建扩展类型的属性。所有属性都使用字符串定义。在 Main 方法中再加入如下代码：

```
soet.CLSID = "VegSOE.VegUtilsSOE";
soet.Description = "Veg Utilities Server Object Extension";
soet.Name = "VegUtilitiesSOE";
```

一旦设置好服务器对象扩展的属性，可以使用 IServerObjectAdmin2 接口的 AddExtensionType 方法，将其加入到一服务器对象类中。代码如下：

```
soa.AddExtensionType("MapServer", soet);
Console.WriteLine("在 ArcGIS 服务器上注册服务器对象扩展成功");
Console.ReadLine();
```

编译并运行 RegisterSOE 工程。如果注册成功，将在控制台窗口中显示“在 ArcGIS 服务器上注册服务器对象扩展成功”一行文本信息。

在服务器对象管理器计算机的 ArcGIS 安装目录下的 server\system 路径中，存在名为 ServerTypeExt.dat 的文件。注册后，会在该文件中加入如下内容：

```
<types>
    <ServerObjectType>
        <Name>MapServer</Name>
```

```
        <ExtensionTypes>
            <ExtensionType>
                <Name>VegUtilitiesSOE CSharp</Name>
                <DisplayName></DisplayName>
                <CLSID>VegSOECSharp.VegUtilsSOE</CLSID>
                <Description>Veg Utilities Server Object Extension
                </Description>
            </ExtensionType>
        </ExtensionTypes>
    </ServerObjectType>
</types>
```

9.3.6　在地图服务上应用服务器对象扩展

配置地图服务应用服务器对象扩展最简单的方法是使用 ArcCatalog。

打开 ArcCatalog，创建一个与 ArcGIS 服务器的管理连接。打开 Yellowstone 地图服务的属性配置对话框，切换到 Capabilities 选项卡中。选择 VegUtilitiesSOE 扩展。在该选项卡的 Properties 部分，将图层属性设置为 Vegetation，字段属性设置为 PRIMARY_。设置如图 9.10 所示。

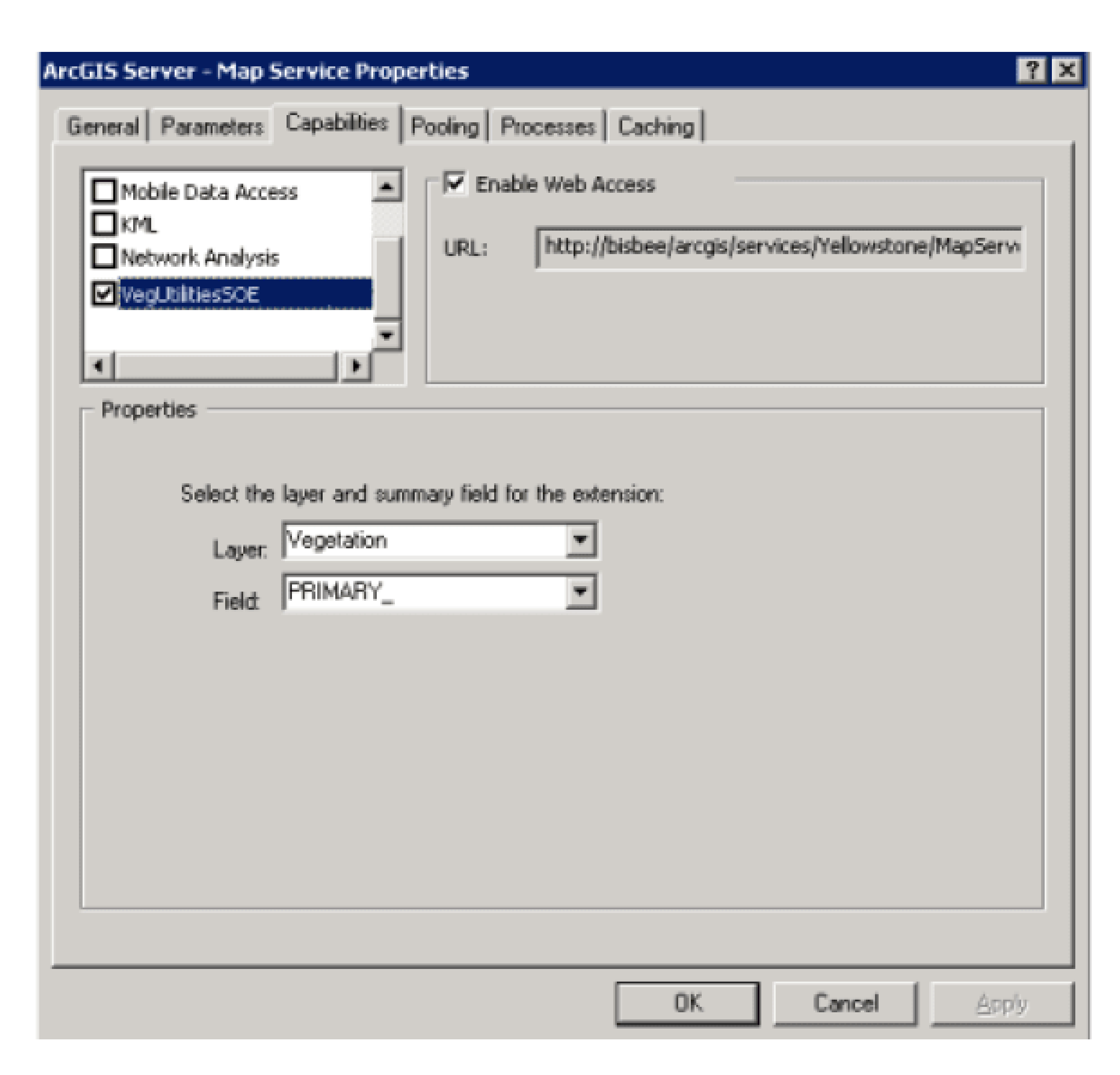


图 9.10　在地图服务上应用服务对象扩展

9.3.7　在 Web 应用程序中访问服务器对象扩展

在解决方案中增加名为 VegSOEWebApp 的 Web 站点。

在 Default.aspx 页面中加入一地图资源管理器控件、一地图控件、一工具栏控件、一 Toc 控件、一标签控件、一文本框控件、一复选框控件、一 IMG 控件以及一个 Div 控件，并在 Div 控件中加入一个 GridView 控件。在地图资源管理器控件中加入 Yellowstone 地图资源。在工具栏控件中加入“放大”、“缩小”、“漫游”工具以及一个自定义工具。该页面设计期间的视图完全同于 9.2.3

节的 COMTest 的 Default.aspx 页面。

在 Default.aspx 页面的<head>与</head>部分，加入如下 JavaScript 内容，用于获取用户在文本框与复选框提供的值：

```
<script language="javascript" type="text/javascript">
    function SetCheckBox(id) {
      if (id == 'CheckBox1')
      {
          document.getElementById('CheckBox1').value = "";
          document.getElementById('CheckBox1').value =
          document.getElementById('CheckBox1').checked;
      }
    }

    function CustomLoad() {
      var customform = document.forms[0];
      var                          original value                          =
customform.elements["ESRIWebADFHiddenFields"].value;
      var new value = original value + ",TextBox1,CheckBox1";
      customform.elements["ESRIWebADFHiddenFields"].value = new value;
    }
</script>
```

当用户选择复选框时调用 SetCheckBox 函数。CustomLoad 函数用于修改一隐藏的文本框元素（ESRIWebADFHiddenFields），Web ADF 用该元素在浏览器端存储一些结果。

ESRIWebADFHiddenFields 文本框元素是 Web 页面在运行期间写入的。它的值是用逗号分割的页面中一组元素的 id，如下所示：

```
<input type="hidden" name="ESRIWebADFHiddenFields"
       id="ESRIWebADFHiddenFields"
       value="maxx,  maxy,  minx,  miny,  coords,  Map1 mode,  control,
Toolbar1_Group_current_tool," />
```

当 Web ADF 生成一回发（回调或整个页面的回发）时，display_dotnetadf.js 文件中的 createClientPostBackQueryString 函数对隐藏文本框中的元素 id 进行循环，加入每个字段的参数与值。为充分利用该功能，我们在其中加入文本框与复选框的 id。在自定义工具的实现代码中将解析这些值。

CustomLoad 函数需要在页面加载时便执行，因此需要设置页面的 body 标签的代码，如下所示：

```
<body onload="CustomLoad()">
```

在工程中加入 ESRI.ArcGIS.Server、ESRI.ArcGIS.ADF.ArcGISServer、ESRI.ArcGIS.Carto、ESRI.ArcGIS.ADF.Web.DataSources.ArcGISServer、ESRI.ArcGIS.System、ESRI.ArcGIS.Geometry 与 ESRI.ArcGIS.Display 程序集的应用。

在 App_Code 目录下加入名为 `VegTool` 的类。该类用于实现自定义工具，在该工具中调用服务器对象扩展计算缓冲区范围内各种植被的面积。在该类的头部加入如下命名空间的引用：

```
using ESRI.ArcGIS.ADF.Web.UI.WebControls;
using ESRI.ArcGIS.ADF.Web.UI.WebControls.Tools;
```

```
using ESRI.ArcGIS.ADF.ArcGISServer;
using ESRI.ArcGIS.ADF.Web.DataSources;
using ESRI.ArcGIS.ADF.Web.DataSources.ArcGISServer;
using ESRI.ArcGIS.Carto;
using ESRI.ArcGIS.Geodatabase;
using ESRI.ArcGIS.Server;
using ESRI.ArcGIS.Geometry;
using VegSOEInterfaces;
```

修改 VegTool 类的声明代码：

```
public class VegTool : IMapServerToolAction
```

利用集成开发环境的自动生成代码功能，生成 IMapServerToolAction 接口实现的代码框架。在 ServerAction 方法中首先加入如下代码：

```
ESRI.ArcGIS.ADF.Web.UI.WebControls.Map mapctrl = args.Control
    as ESRI.ArcGIS.ADF.Web.UI.WebControls.Map;

System.Web.UI.Page page = mapctrl.Page;
string cbxvalue = String.Empty;
string tbxvalue = String.Empty;
string callbackArgs = String.Empty;
System.Collections.Specialized.NameValueCollection keyValColl = null;

if (page.IsCallback)
{
    callbackArgs = page.Request.Params["__CALLBACKPARAM"];
    keyValColl =
        CallbackUtility.ParseStringIntoNameValueCollection(callbackArgs);
}
else
{
    keyValColl = page.Request.Params;
}

tbxvalue = keyValColl["TextBox1"];
cbxvalue = keyValColl["CheckBox1"];
```

在上面的代码中，如果工具启动的是一回调，那么 HTTP 请求中的__CALLBACKPARAM 参数包含一参数/值对，如果启动的是一整个页面的回发，那么参数/值对包含在 HTTP 请求中。

然后在方法中加入如下代码，用于获取服务器上下文以及将屏幕坐标的点对象转换为地图坐标点对象：

```
MapFunctionality mapfunc =
    (MapFunctionality)mapctrl.GetFunctionality("Yellowstone");
MapResourceLocal mapres = (MapResourceLocal)mapfunc.MapResource;
IServerContext sc = mapres.ServerContextInfo.ServerContext;
IMapServer mapserver = mapres.MapServer;

PointEventArgs pargs = (PointEventArgs)args;
```

```
ESRI.ArcGIS.ADF.Web.Geometry.Point inpt =
    ESRI.ArcGIS.ADF.Web.Geometry.Point.ToMapPoint(
        pargs.ScreenPoint, mapctrl.Extent,
        mapfunc.DisplaySettings.ImageDescriptor.Width,
        mapfunc.DisplaySettings.ImageDescriptor.Height);
IPoint pt = (IPoint)sc.CreateObject("esriGeometry.Point");
pt.X = inpt.X;
pt.Y = inpt.Y;

double distance = 0;

if (!Double.TryParse(tbxvalue, out distance)) {
    distance = 10000;
}
```

接着加入如下代码：

```
IServerObjectExtensionManager soext_manager = mapserver
        as IServerObjectExtensionManager;
IServerObjectExtension soext = soext_manager.FindExtensionByTypeName
("VegUtilitiesSOE");
IVegUtilsSOE vegutils = (IVegUtilsSOE)soext;
IVegResultsSOE vegresults = vegutils.sumVegetationType(pt, distance);
```

如果服务器对象上应用了一服务器对象扩展，那么可以通过 FindExtensionByTypeName 方法返回指定注册名称的服务器对象扩展的引用。得到服务器对象扩展后，便可调用其中的方法。

然后在方法中加入如下高亮度绘制选择要素的代码：

```
IGraphicElements comGraphics = vegresults.ResGraphics;
GraphicElement[] proxyGraphics = (GraphicElement[])
    ESRI.ArcGIS.ADF.ArcGISServer.Converter.ComObjectToValueObject(
    comGraphics, sc, typeof(GraphicElement[]));

RgbColor rgb = new RgbColor();
rgb.Red = 155;
rgb.Green = 0;
rgb.Blue = 0;
rgb.AlphaValue = 255;

SimpleLineSymbol sls = new SimpleLineSymbol();
sls.Style = ESRI.ArcGIS.ADF.ArcGISServer.esriSimpleLineStyle.esriSLSSolid;
sls.Color = rgb;
sls.Width = 0.2;

foreach (ESRI.ArcGIS.ADF.ArcGISServer.PolygonElement pe in proxyGraphics) {
    SimpleFillSymbol sfs = (SimpleFillSymbol)pe.Symbol;
    sfs.Outline = sls;
}

mapfunc.MapDescription.CustomGraphics = proxyGraphics;
```

统计结果显示的代码如下：

```
    GridView gdview = (GridView)mapctrl.Page.FindControl("GridView1");
    string showtable = "'hidden'";
    Boolean displaydiv = false;

    if (!Boolean.TryParse(cbxvalue, out displaydiv)) {
        displaydiv = false;
    }

    if (displaydiv) {
        IRecordSet rs = vegresults.Stats;
        ESRI.ArcGIS.ADF.ArcGISServer.RecordSet value_rs =
               ESRI.ArcGIS.ADF.Web.DataSources.ArcGISServer.Converter.ComObject
ToValueObject(rs, sc, typeof(ESRI.ArcGIS.ADF.ArcGISServer.RecordSet)) as
ESRI.ArcGIS.ADF.ArcGISServer.RecordSet;
        System.Data.DataTable datatable =
            ESRI.ArcGIS.ADF.Web.DataSources.ArcGISServer.Converter.ToDataTable(
value rs);

        gdview.DataSource = datatable;
        gdview.DataBind();

        string returnstring = null;

        using (System.IO.StringWriter sw = new System.IO.StringWriter()) {
            HtmlTextWriter htw = new HtmlTextWriter(sw);
            gdview.RenderControl(htw);
            htw.Flush();
            returnstring = sw.ToString();
        }

        CallbackResult cr = new CallbackResult("div", "griddiv", "innercontent",
returnstring);
        mapctrl.CallbackResults.Add(cr);

        if (datatable.Rows.Count > 1)
            showtable = "'visible'";
    }
```

最后加入如下代码，用于设置 Div 的可见性以及刷新地图：

```
    object[] oa = new object[1];
    string sa = "var griddiv = document.getElementById('griddiv');";
    sa += "griddiv.style.visibility = " + showtable + ";";
    oa[0] = sa;

    CallbackResult cr1 = new CallbackResult(null, null, "javascript", oa);
    mapctrl.CallbackResults.Add(cr1);

    if (mapctrl.ImageBlendingMode == ImageBlendingMode.WebTier) {
```

```
        mapctrl.Refresh();
    }
    else if (mapctrl.ImageBlendingMode == ImageBlendingMode.Browser) {
        mapctrl.RefreshResource(mapres.Name);
    }
```

由于我们在自定义工具中已经实现了查询以及结果显示功能，因此对于_Default 类，要实现的功能就很少了。只需要在其 Page_Load 方法中加入如下代码，设置复选框的客户端 onclick 事件响应函数：

```
    CheckBox1.Attributes.Add("onclick", "ChangeCheckBoxValue()");
```

编译并运行程序。该 Web 程序的界面、使用方式与结果都与 9.2.3 节的 Web 程序完全相同。

第 10 章



GIS Web服务的应用与创建

ArcGIS 提供了两种类型的 Web 服务创建方法，分别是 GIS Web 服务和应用性 Web 服务。

GIS Web 服务提供了一种将 ArcGIS 服务器对象（本地数据源）发布为 ArcGIS Server Web 服务（远程数据源）的 ESRI 标准，GIS Web 服务不用于开发，通常用来发布信息和提供资源。应用性 Web 服务是基于标准 Web 服务建立的应用，需要开发人员使用某种数据源进行开发。

通过本章你将了解到：

10.1 GIS Web 服务的应用
10.2 应用性 Web 服务的创建与使用

10.1 GIS Web 服务的应用

ArcMap 等桌面应用程序可以直接使用 GIS Web 服务的资源而不用进行任何开发，另外 Web ADF 控件和公有 API 也可以使用 GIS Web 服务资源。由于 GIS Web 服务基于标准 Web 服务，它可以作为传统 Web 服务来使用，ArcGIS Server 提供了 SOAP API 进行相关的开发。

可以通过本地与远程两种方式来访问 ArcGIS Server 服务。在应用程序中，本地连接是通过 ArcObjects 来连接服务器对象管理器的。而远程连接是通过本地对象连接 Web 服务端点。本地对象意味着是本地开发环境（例如.NET 或 Java）管理的对象。本地连接使用 ArcObjects API 与 SOAP API，操作服务，也就是服务器对象与服务器上下文。服务（服务器对象）的状态的改变只能使用 ArcObjects API 来实现。远程连接只能使用 SOAP API 来操作服务，所以所有与服务的交互都是无状态的。通常，在客户应用程序中 ArcGIS Server 的连接可以总结如下，如图 10.1 所示：

（1）远程连接总是使用 SOAP API，使用 Web 服务端点的 URL 定义连接。远程连接可用于局域网内部的应用，也可用于 Internet 网外部的应用。

（2）本地连接通常总是使用 ArcObjects API，也可以使用 SOAP API，这需要根据客户应用程序的需要决定。本地连接只能用于局域网内部的应用。

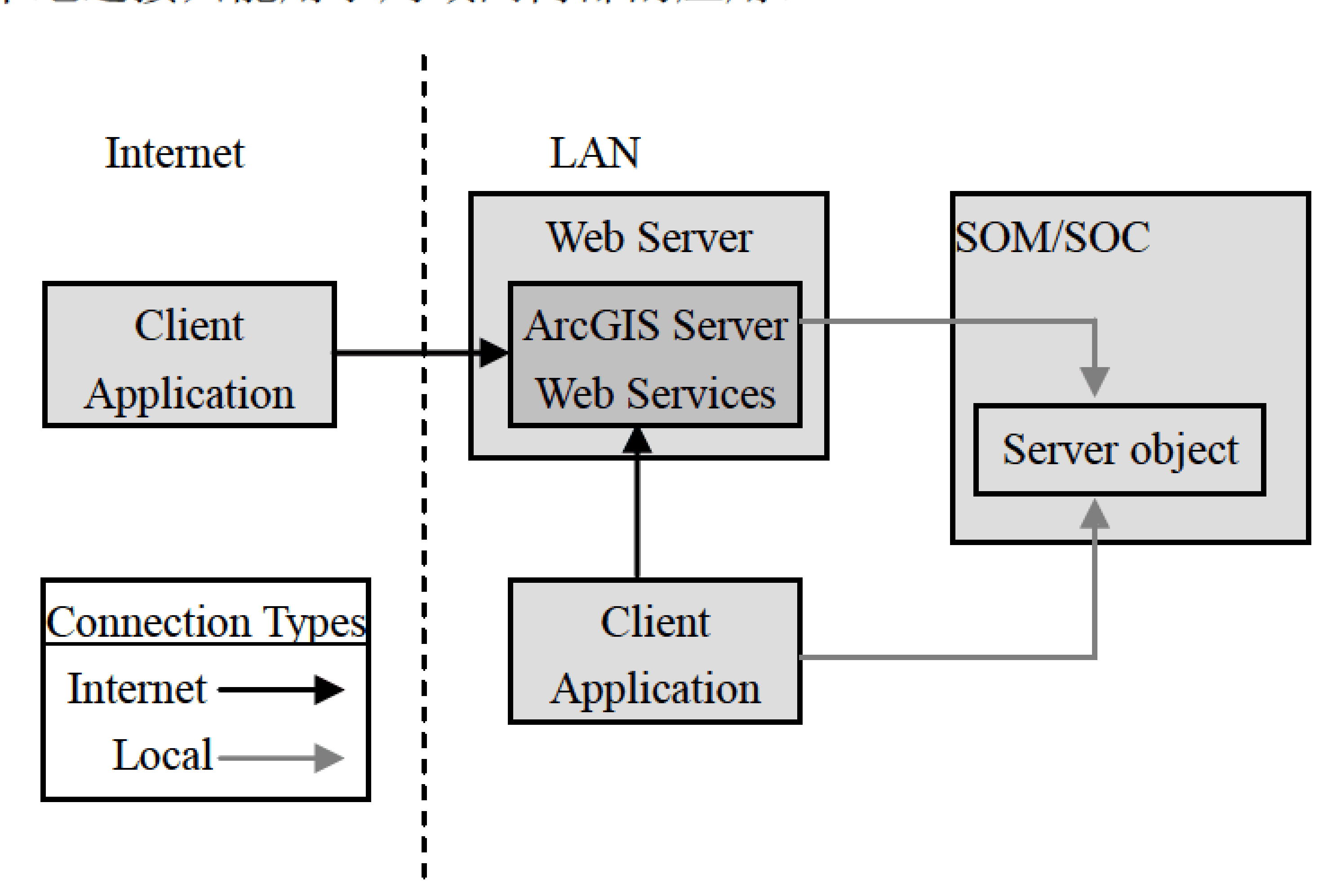


图 10.1 本地连接与远程连接的区别

在桌面应用程序中，例如 ArcMap、ArcCatalog 与 ArcGIS Explorer，可以使用两种类型的连接访问事先创建的 ArcGIS Server 服务。Web ADF 与 ArcGIS Server 管理器也不需要显式定义使用的 API，便可以使用 ArcGIS Server 服务。对于应用程序开发人员必须清楚 ArcGIS Server 服务的功能与限制。因此，使用 SOAP API 需要开发人员掌握一定的 Web 服务标准与规范，以及 ArcGIS Server 服务的功能。

.NET 或 Java 开发环境都提供了 SOAP 工具，创建本地的值对象以及代理类。WSDL 提供了创建代理类与值对象需要的所有信息。客户应用程序从服务器得到 WSDL，创建代理类与值对象的过程如图 10.2 所示。

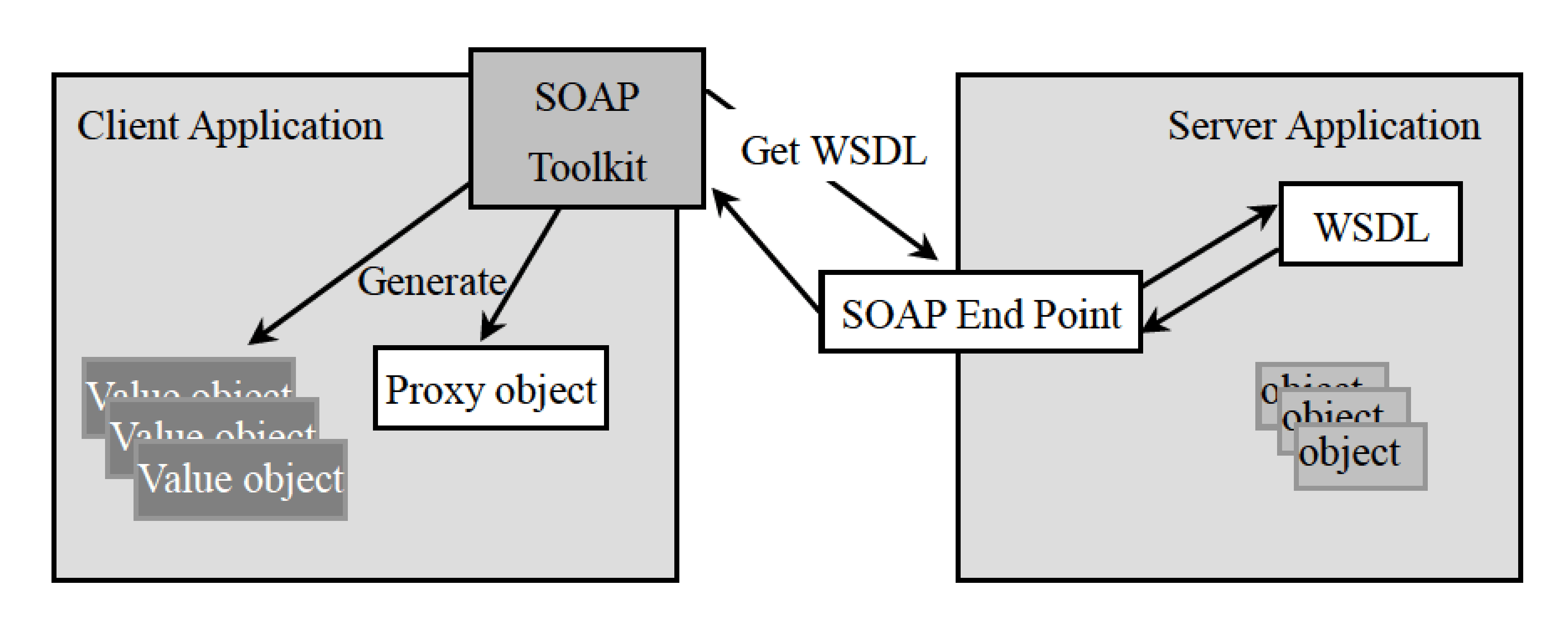


图 10.2　从服务器中获取 WSDL 并创建代理类与值对象的过程

值对象是存储值与属性的简单对象。根据 WSDL，可以是许多不同类型的值对象。另一方面，每个 SOAP 服务类型只有一个代理类。代理类同时包含属性与方法。方法用于创建与提交 SOAP 请求，并返回 SOAP 响应。代理类使用值对象来创建（序列化）SOAP 请求。代理类同时也反序列化 SOAP 响应，构造值对象，如图 10.3 所示。因此，在富客户端（Rich Client）对象的环境中，由于使用本地值对象存储属性并管理 SOAP 请求与响应，使用 SOAP 服务相对要容易得多。

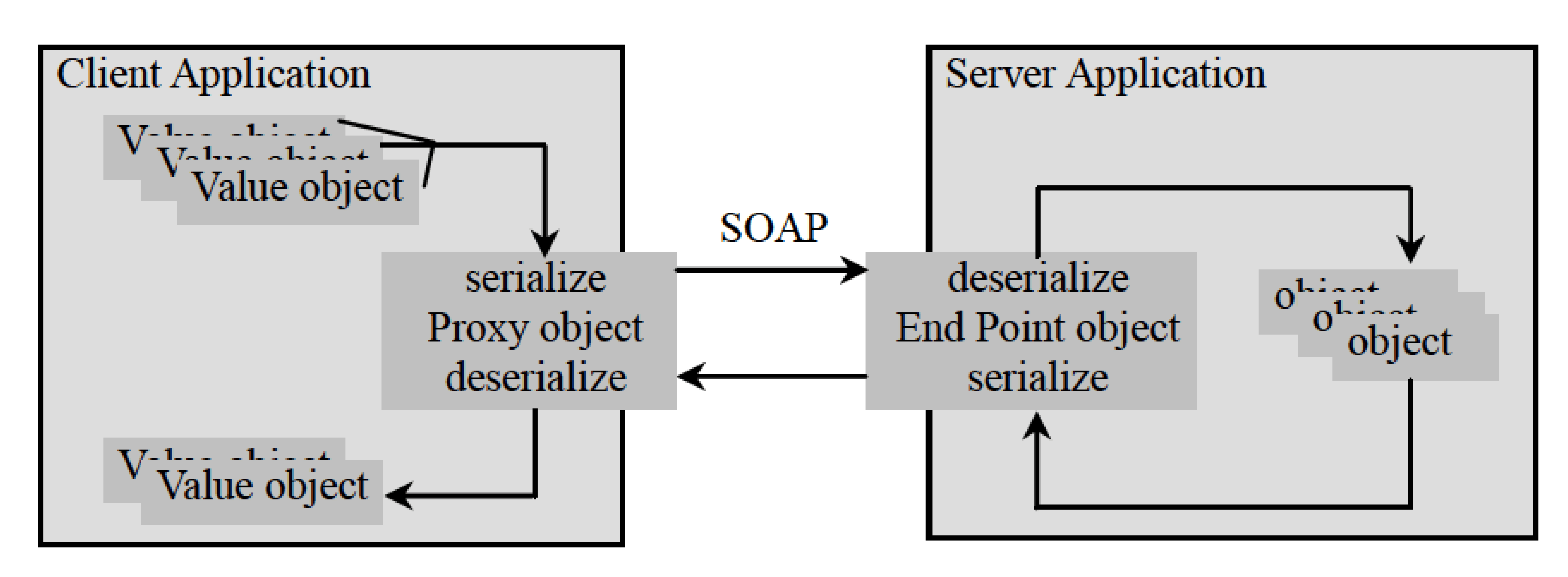


图 10.3　序列化请求与反序列化响应

对于使用 SOAP API 访问 ArcGIS Server Web 服务的开发人员，根据连接类型与客户端组件，可以使用两种方法开发 ArcGIS Server 应用程序，如图 10.4 所示。

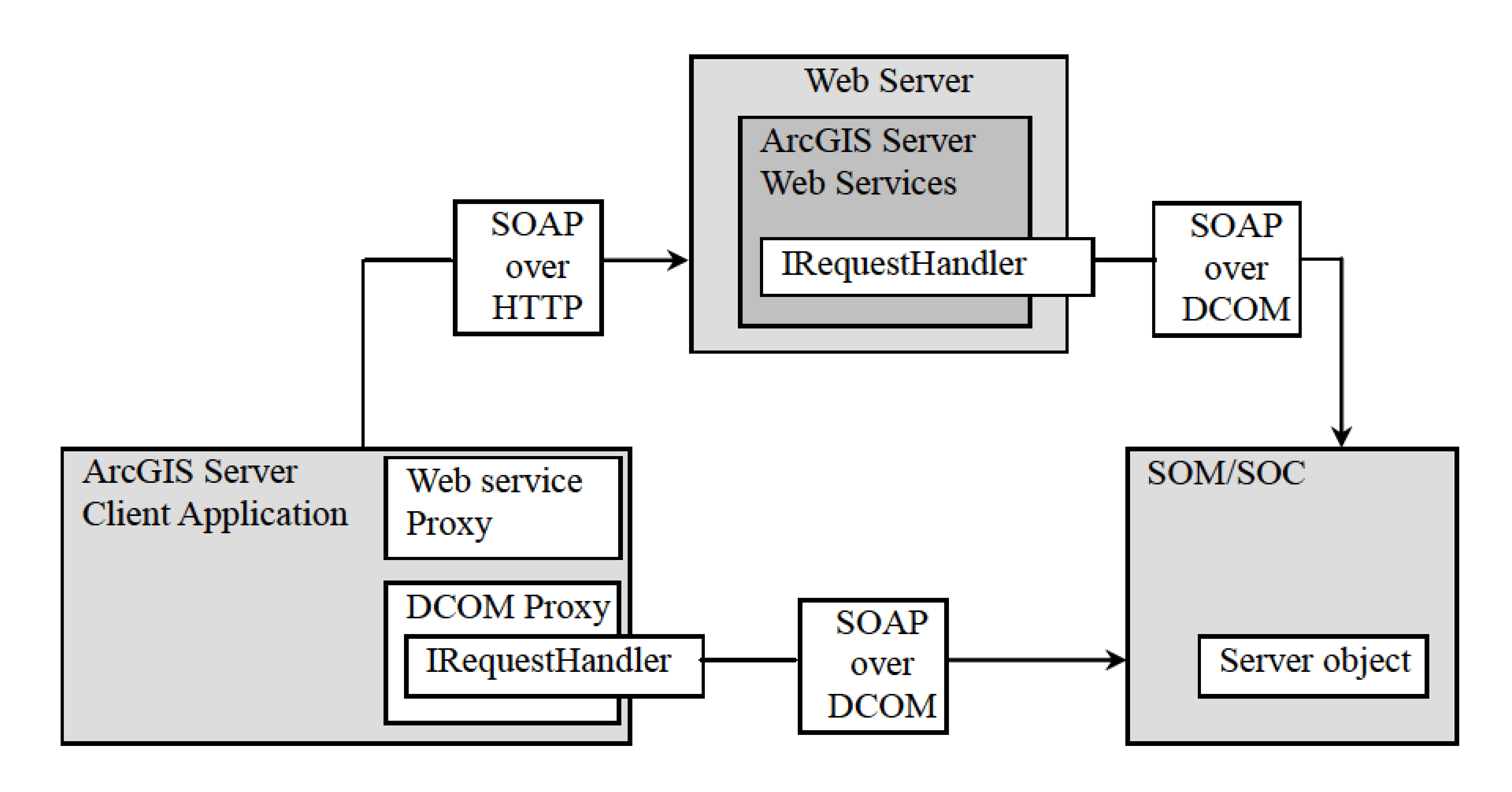


图 10.4　通过 SOAP API 访问 Web 服务的两种方式

如果可以使用 Web ADF，则可使用 ESRI.ArcGIS.ADF.ArcGISServer 程序集中事先创建的 ArcGIS SOAP 代理类与值对象。对于支持 SOAP 的每个服务器类型，提供了两个代理，分别是 Web 服务代理与 DCOM 代理。Web 服务代理使用基于 HTTP 的 SOAP 访问 ArcGIS Server Web 服务。DCOM 代理使用基于 DCOM 的 SOAP，通过 IRequestHandler 接口访问服务器对象，因此需要 ArcGIS Server 本地连接。在这两种情况下，代理类使用的值对象在同一类库与命名空间中。

如果不能使用 Web ADF，则可使用.NET 开发包中的 SOAP 工具，从 ArcGIS Server 的 Web 服务端点，生成 Web 服务的代理类与值对象。由于在客户端没有 ArcObjects COM 代理类，因此不能访问远程的 COM 对象，所以也就不能使用基于 DCOM 的 SOAP。使用这种方式，不需要在客户端安装任何 ESRI 的产品。

对于 ArcGIS Server 的.NET 版本，ArcGIS Server Web 服务就是一 ASP.NET Web 应用程序，包含了一 HTTP 处理程序。HTTP 处理程序内部使用 IServiceCatalogAdmin 与 IRequestHandler 接口向外提供基于 HTTP 的 SOAP API。ArcGIS Server Web 服务应用程序，管理通过 ArcObjects 与 SOAP API 的交互，因此开发人员不需要使用 ArcObjects。这意味着客户端不需要 ArcObjects COM 代理。事实上，使用 ArcGIS Server Web 服务的客户端不需要安装 ESRI 软件。

ArcGIS Server Web 服务使用如下格式的 URL：

http://<Web Server Hostname>/<ArcGIS Instance>/services/<ServiceName>/<ServiceType>

- <Web Server Hostname>是 Web 服务应用程序发布的 Web 服务器的主机名或 IP 地址；
- <ArcGIS Instance>是包含一系列服务的虚拟路径的名称。每个服务器对象管理器有一个名称，在 ArcGIS Server 后安装过程中指定的名称；
- <ServiceName>是某一 ArcGIS Server 服务的名称；
- <ServiceType>是服务器对象或服务器对象扩展类型。ArcGIS Server 包含了一组标准的服务类型，例如 MapServer、GeocodeServer 等。每个类型必须提供一 WSDL，并实现 IRequestHandler 接口。

服务器对象或扩展的 WSDL 提供了通过 SOAP 与 Web 服务访问的必要信息。在服务的 URL 后加上“?wsdl”可访问服务器对象或扩展的的 WSDL。

每个服务类型有一代理类。SOAP 代理定义了 ArcGIS Server 服务的功能。正如前面所介绍的，SOAP API 与服务交互是无状态的，因此代理类提供方法初始化对服务的无状态的请求并返回结果。SOAP 代理的功能反映了服务器对象实现的无状态的 ArcObjects 接口。例如，MapServer SOAP 代理中的方法同于 ArcObjects MapServer 对象实现的无状态 ArcObjects 接口（IMapServer 与 ITiledMapServer）的方法。ArcGIS Server SOAP 代理调用方法与返回结果时，使用值对象。因此，SOAP 值对象与 ArcObjects 类型使用相同名称的方法，但是不同的对象类型。值对象是本地的.NET 对象，而服务器对象接口引用的是在 GIS 服务器上的 COM 对象。例如，MapServer SOAP 代理与 IMapServer ArcObjects 接口都包含名为 ExportMapImage 方法。但是，SOAP 代理需要两个输入参数，分别是 MapDescription 与 ImageDescription 类型的.NET 对象，而 IMapServer ArcObjects 接口要求的两个参数分别是 IMapDescription 与 IMageDescription 接口引用的在 GIS 服务器上的 COM 对象。MapServer 代理返回一个本地的.NET 的 MapImage 值对象，而 IMapServer 返回的是一 IMapImage 接口引用的在 GIS 服务器上的 COM 对象。

Web ADF 同时包含了通过 Internet（Web 服务）以及本地（DCOM）连接的 SOAP 代理类。.NET

提供了 SOAP 工具包 wsdl.exe，用于根据一 WSDL 创建代理类与值对象。

下面的代码演示了如何利用 SOAP 代理类初始化 ArcGIS Server 地图服务。

（1）通过 Web 服务代理实现，代码如下：

```
    string endpoint =
       "http://MyWebServer/arcgis/services/MyMapServiceName/MapServer";
    ESRI.ArcGIS.ADF.ArcGISServer.MapServerProxy mapserver =
       new ESRI.ArcGIS.ADF.ArcGISServer.MapServerProxy(endpoint);
```

（2）通过 DCOM 代理（需要 ESRI.ArcGIS.ADF.dll 与 ESRI.ArcGIS.ADF.Connection.dll 连接程序集）实现，代码如下：

```
    ESRI.ArcGIS.ADF.Identity id = new ESRI.ArcGIS.ADF.Identity(username, password,
domain);
    ESRI.ArcGIS.ADF.ArcGISServer.MapServerDcomProxy mapserver_dcom =
       (MapServerDcomProxy) MapServerDcomProxy.Create(SOMname, servicename, id);
    // 通过 DCOM 代理调用方法
    mapserver_dcom.Dispose();
```

（3）动态实现，代码如下：

```
    wsmap.AMapServiceName_MapServer mapserver =
       new wsmap.AMapServiceName_MapServer();
    mapserver.Url =
       "http://MyWebServer/arcgis/services/MyMapServiceName/MapServer";
```

对于 ArcGIS Server 服务，只需要针对每种服务类型生成一次代理类与值对象。一旦生成了代理类与值对象，即使代理类名称中包含了原服务的名称，也可用于访问其他同类型的 ArcGIS Server Web 服务。

下面我们通过一个实例来演示如何使用 ArcGIS Server Web 服务目录以及 MapServer Web 服务。

在 Visual Studio 中创建一个名为 MapServerBrowser 的 Windows 应用程序。

要在应用程序中访问 Web 服务目录以及 MapServer Web 服务提供的方法与对象，需要在应用程序中加入它们的 Web 引用。

在解决方案浏览器中，选择工程右键菜单的 Add Web Reference 命令，打开如图 10.5 所示的 Add Web Reference 对话框。在该对话框的 URL 文本框中输入 Web 服务目录的 WSDL 地址，即"http://localhost/arcgis/services?wsdl"，然后回车，或选择 Go。一旦找到 Web 服务，在 Web reference name 文本框中输入 WebCatalog，然后选择 Add Reference。集成开发环境将利用 SOAP 工具包创建该 Web 服务的代理类与值对象。

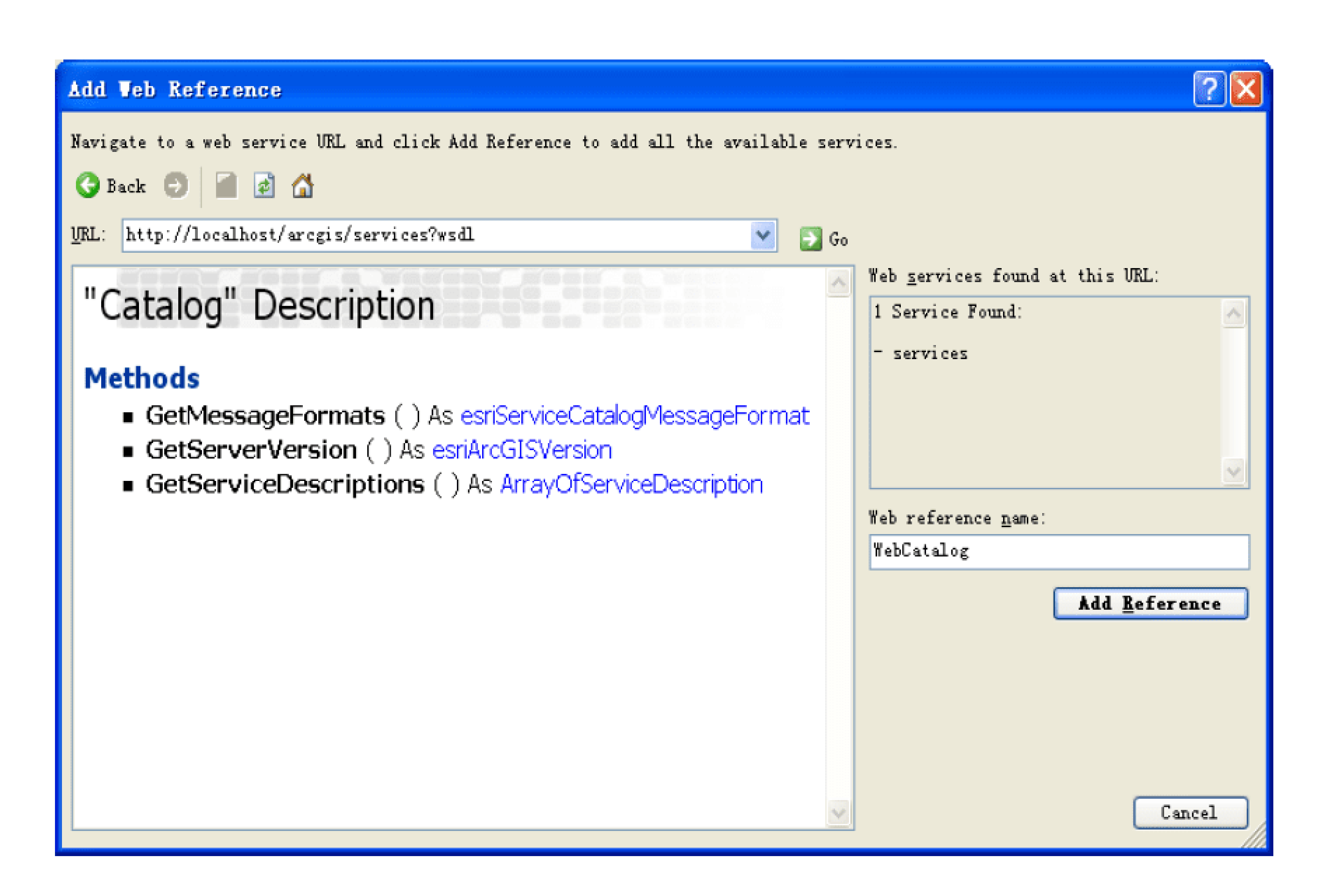


图 10.5 Add Web Reference 对话框

通过同样的方式，生成 http://localhost/arcgis/services/NorthAmericaMap/MapServer?wsdl（NorthAmericaMap 地图服务）URL 地址的 Web 引用，将名称命名为 NorthAmericaMapWS。

这时可在解决方案浏览器中发现新增加了一个名为 Web References 的文件夹。在该文件中包含了 WebCatalog 与 NorthAmericaMapWS 两 Web 引用。双击其中一个，打开对象浏览器，如图 10.6 所示，可以查看其中的方法。

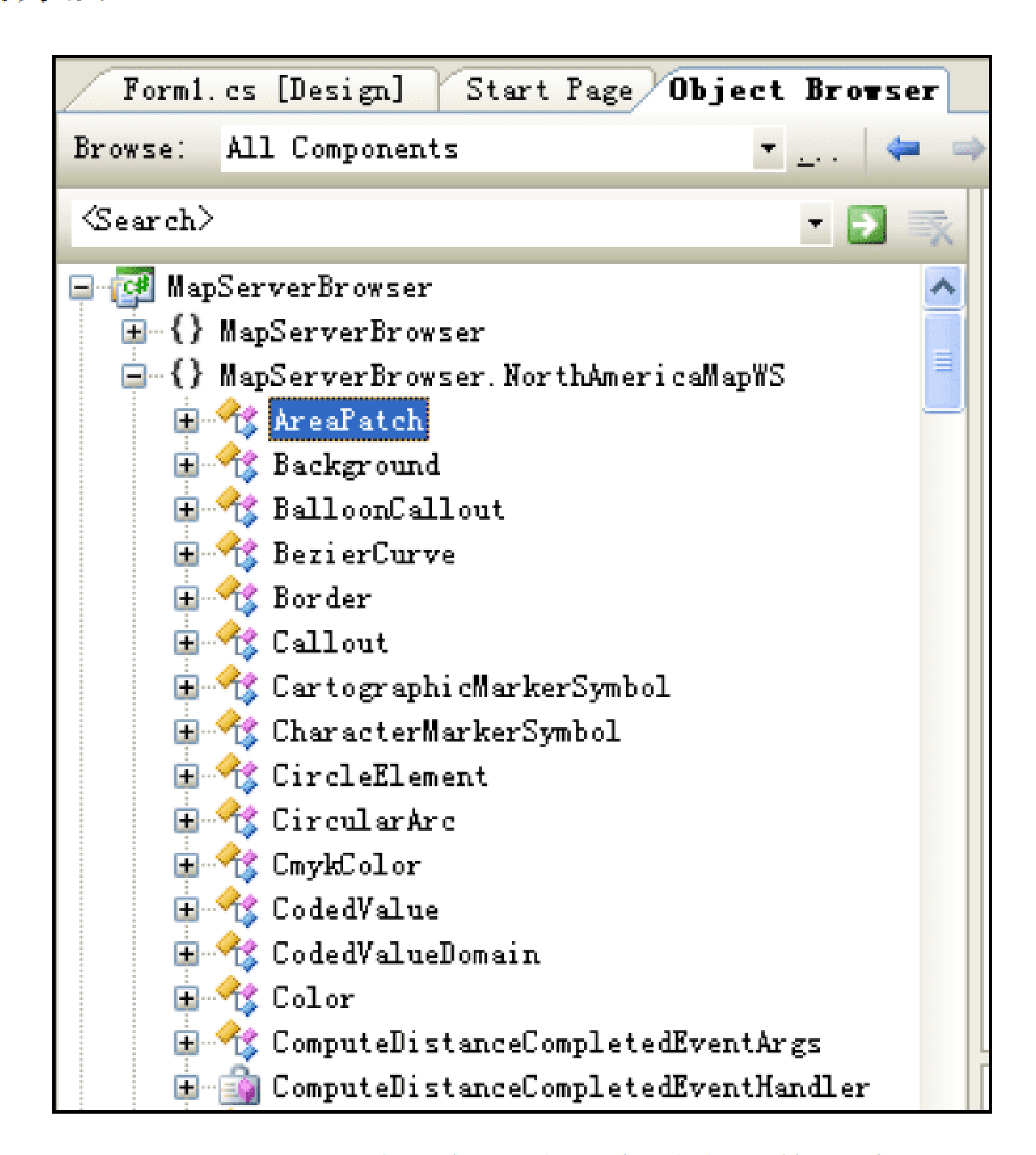


图 10.6 通过对象浏览器查看类及其方法

加入 Web 服务的引用后，我们便可使用其中的类与方法，访问与操作 ArcGIS Server Web 服务。

在 Form1 窗体中加入四个标签控件、三个下拉列表框空间、一文本框控件、一按钮控件、一图片框（PictureBox）控件与一工具条（ToolStrip）控件。

通过工具条控件的 Items 属性，打开 Items Collection Editor 对话框。在该对话框中加入“放大”、“缩小”、“全图”与“漫游”工具，如图 10.7 所示。

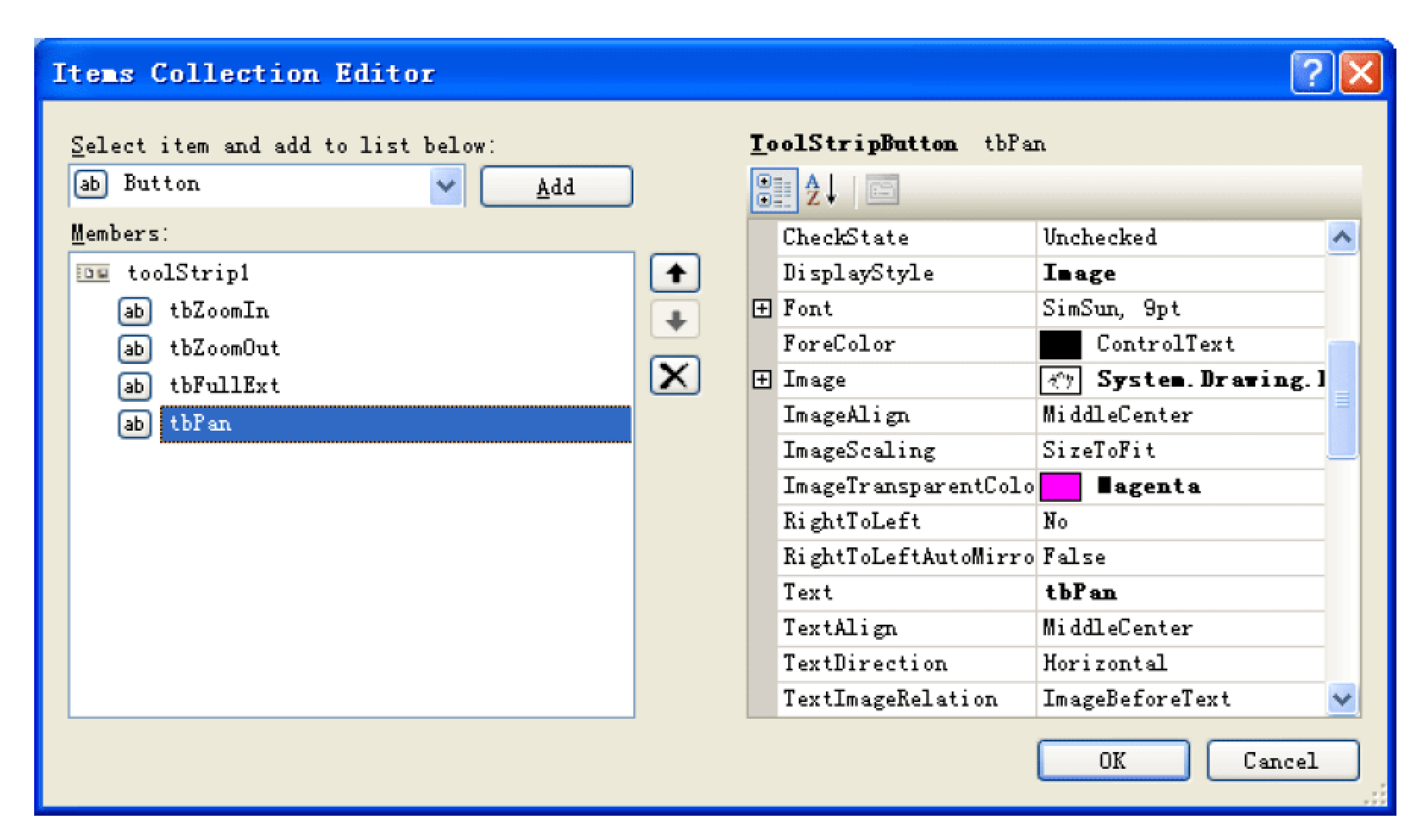


图 10.7 通过 Items Collection Editor 对话框加入工具

Form1 窗体在设计期间的视图如图 10.8 所示。

图 10.8 窗体 Form1 的设计视图

切换到 Form1.cs 的代码文件中，首先在类中加入如下字段：

```
private string m sSelectedMap;
private NorthAmericaMapWS.MapDescription m sMapDesc;
private string m sDataFrame;
private double startX;
```

```
private double startY;
private int startDragX;
private int startDragY;
private int deltaDragX;
private int deltaDragY;
private NorthAmericaMapWS.ImageDisplay idisp;
private System.Drawing.Image pImage;
private Boolean ispanning;
```

创建按钮的 Click 事件响应方法。在该方法中，需要完成连接 Web 服务目录，得到一组 MapServer Web 服务，并加入到 Web 服务下拉列表框。

由于需要跨进程 SOAP 调用，该操作可能需要花费几秒时间来执行，因此需要在应用程序中向用户显示正在运行的状态。这可以通过显示一个等待光标来实现。在按钮的 Click 事件响应方法中首先加入代码：

```
this.Cursor = Cursors.WaitCursor;
cboMapServer.Items.Clear();
```

然后加入如下代码，实现查询：

```
try {
    WebCatalog.Catalog sc = new WebCatalog.Catalog();
    sc.Url = txtServer.Text;
    WebCatalog.ServiceDescription[] wsdesc = sc.GetServiceDescriptions();
    WebCatalog.ServiceDescription sd = null;

    for (int i = 0; i < wsdesc.Length; i++) {
        sd = wsdesc[i];
        if (sd.Type == "MapServer") {
            cboMapServer.Items.Add(sd.Url);
        }
    }

    cboMapServer.Enabled = true;
    this.Cursor = Cursors.Default;
}
catch (Exception exception) {
    this.Cursor = Cursors.Default;
    MessageBox.Show(exception.Message, "An error has occurred");
}
```

在上述代码中，首先创建一个 WebCatalog.Catalog 对象，然后设置该对象的 URL，然后调用该对象的 GetServiceDescriptions 方法，得到 ServiceDescription 对象数组。然后通过一个循环得到 MapServer Web 服务的 URL，并加入到 Web 服务下拉列表框（cboMapServer）中。

下面要实现的是得到指定的地图服务的数据集名称数组。

创建 Web 服务下拉列表框（cboMapServer）的选择索引改变（SelectedIndexChange）事件的响应方法。在其中加入如下代码：

```
m_sSelectedMap = cboMapServer.Text;
try {
```

Web GIS

```
    NorthAmericaMapWS.NorthAmericaMap_MapServer map =
         new NorthAmericaMapWS.NorthAmericaMap_MapServer();
    map.Url = m sSelectedMap;

    m sDataFrame = map.GetDefaultMapName();

    // 得到数据集名称，并填充到下拉列表框中
    cboDataFrame.Items.Clear();
    for (int i = 0; i < map.GetMapCount(); i++) {
       cboDataFrame.Items.Add(map.GetMapName(i));
    }
    cboDataFrame.Text = m sDataFrame;

    // 控制控件的状态
    cboDataFrame.Enabled = true;
    cboBookMark.Enabled = true;
    IEnumerator benum = toolStrip1.Items.GetEnumerator();
    ToolStripButton btn = null;
    while (benum.MoveNext()) {
       btn = (ToolStripButton)benum.Current;
       btn.Enabled = true;
    }
}
catch (Exception exception) {
    MessageBox.Show(exception.Message, "An error has occurred");
}
```

在上面的方法中，首先创建一个地图服务器实例，然后调用其 GetDefaultMapName 方法得到当前选择的数据集的名称。然后通过循环，将地图中的数据集名称加入到数据集下拉列表框（cboDataFrame）中。

下面要实现的是响应数据集改变的代码。创建数据集下拉列表框（cboDataFrame）的选择索引改变（SelectedIndexChanged）事件响应方法。在其中加入如下代码：

```
m sDataFrame = cboDataFrame.Text;

try {
    // 从地图服务器查询选择的数据集的信息
    NorthAmericaMapWS.NorthAmericaMap MapServer map =
        new NorthAmericaMapWS.NorthAmericaMap MapServer();
    map.Url = m sSelectedMap;

    // 得到书签，并填充书签下拉列表框
    NorthAmericaMapWS.MapServerInfo mapi = map.GetServerInfo(m_sDataFrame);
    NorthAmericaMapWS.MapServerBookmark[] pMSBookMarks = mapi.Bookmarks;

    cboBookMark.Items.Clear();
    cboBookMark.Items.Add("<Default Extent>");
    NorthAmericaMapWS.MapServerBookmark pMDBook;
    for (int j = 0; j < pMSBookMarks.Length; j++) {
       pMDBook = pMSBookMarks[j];
```

```
            cboBookMark.Items.Add(pMDBook.Name);
        }
        cboBookMark.SelectedItem = "<Default Extent>";
    }
    catch (Exception exception) {
        MessageBox.Show(exception.Message, "An error has occurred");
    }
```

在上面的代码中，首先创建一个地图服务器对象，并设置该对象的 Url 属性，然后调用其 GetServerInfo 方法得到指定数据集的 MapServerInfo 对象，从该对象中得到一个空间书签数组。然后利用一个循环，将书签的名称加入到数据下拉列表框中。

前面的代码是通过 Web 服务目录以及 MapServer Web 服务的方法，得到它们的信息，并填充到下拉列表框中。

下面要完成的就是在图片框中绘制地图。我们将绘制地图的代码封装在如下辅助方法中：

```
    private void DrawMap(ref NorthAmericaMapWS.MapDescription pMapDescriptoin)
```

该方法需要的参数是 MapDescription 类型的对象，当需要输出地图时，地图服务器利用该地图描述对象中的属性，绘制地图，而不改变运行的地图服务器对象。这些属性包括绘制的范围以及图层的可见性等。可见地图描述的目的是实现无状态地应用地图服务器，因此需要在应用程序中保存这些状态的信息。

在 DrawMap 方法中加入如下代码：

```
    NorthAmericaMapWS.NorthAmericaMap_MapServer map =
        new MapServerBrowser.NorthAmericaMapWS.NorthAmericaMap_MapServer();
    map.Url = m sSelectedMap;

    // 设置输出地图的图像描述
    NorthAmericaMapWS.ImageType it = new NorthAmericaMapWS.ImageType();
    it.ImageFormat = NorthAmericaMapWS.esriImageFormat.esriImageJPG;
    it.ImageReturnType                                                    =
NorthAmericaMapWS.esriImageReturnType.esriImageReturnMimeData;

    idisp = new NorthAmericaMapWS.ImageDisplay();
    idisp.ImageHeight = 400;
    idisp.ImageWidth = 552;
    idisp.ImageDPI = 150;

    NorthAmericaMapWS.ImageDescription pID =
     new NorthAmericaMapWS.ImageDescription();
    pID.ImageDisplay = idisp;
    pID.ImageType = it;

    NorthAmericaMapWS.MapImage pMI = map.ExportMapImage(pMapDescriptoin, pID);
    System.IO.Stream pStream = new System.IO.MemoryStream((byte[])pMI.ImageData);
    pImage = Image.FromStream(pStream);

    pictureBox1.Image = pImage;
    pictureBox1.Refresh();
```

在上面的代码中，首先创建一地图服务器对象，并指定其 Url 属性。然后创建一个图像描述对象。接着调用地图服务器对象的 ExportMapImage 方法绘制地图，并将其转换为一.NET 的图像对象。最后将该图像对象赋予图片框控件。

当用户从书签下拉列表框中选择一个书签时，图片框控件中将显示选择书签的范围中的地图。创建书签下拉列表框（cboBookMark）的选择索引改变事件响应方法。在其中加入如下代码：

```
this.Cursor = Cursors.WaitCursor;

try {
    NorthAmericaMapWS.NorthAmericaMap MapServer map =
        new MapServerBrowser.NorthAmericaMapWS.NorthAmericaMap MapServer();
    map.Url = m sSelectedMap;

    NorthAmericaMapWS.MapServerInfo mapi = map.GetServerInfo(m sDataFrame);
    NorthAmericaMapWS.MapDescription pMapDescription;

    // 如果选择默认的范围，则从地图描述中得到绘制范围
    if (cboBookMark.Text == "<Default Extent>") {
        pMapDescription = mapi.DefaultMapDescription;
        m sMapDesc = pMapDescription;
        DrawMap(ref pMapDescription);
        this.Cursor = Cursors.Default;
        return;
    }

    pMapDescription = m sMapDesc;

    // 查询选择的书签
    NorthAmericaMapWS.MapServerBookmark[] pMSBookMarks = mapi.Bookmarks;
    NorthAmericaMapWS.MapServerBookmark pMDBook = null;

    for (int i = 0; i < pMSBookMarks.Length; i++) {
        pMDBook = pMSBookMarks[i];
        if (pMDBook.Name == cboBookMark.Text)
            break;
    }

    // 将地图描述的范围属性设置为书签的范围
    NorthAmericaMapWS.MapArea pMA = pMDBook;
    pMapDescription.MapArea = pMA;

    // 保存地图描述
    m sMapDesc = pMapDescription;
    DrawMap(ref pMapDescription);

    this.Cursor = Cursors.Default;
}
catch (Exception exception) {
    this.Cursor = Cursors.Default;
```

```
        MessageBox.Show(exception.Message, "An error has occurred");
    }
```

下面要实现的就是“放大”、“缩小”、“全图”以及“漫游”工具。创建工具条控件的工具单击事件（ItemClicked）响应方法。在其中加入如下代码框架：

```
    switch(toolStrip1.Items.IndexOf(e.ClickedItem)){
      case 0: // 放大
        break;
      case 1: // 缩小
        break;
      case 2: // 全图
        break;
      case 3: // 漫游
        break;
    }
```

对于“放大”工具，需要先得到地图服务对象，设置其 Url 属性，从 m_sMapDesc 中得到当前地图描述信息，缩小其范围，然后调用 DrawMap 方法绘制地图，最后使用新的范围更新 m_sMapDesc 字段。代码如下：


```
    this.Cursor = Cursors.WaitCursor;
    try {
        NorthAmericaMapWS.NorthAmericaMap MapServer map =
           new MapServerBrowser.NorthAmericaMapWS.NorthAmericaMap MapServer();
        map.Url = m_sSelectedMap;

        NorthAmericaMapWS.MapDescription pMapDescription = m_sMapDesc;

        // 得到当前范围并缩小该范围，然后将该新的范围设置到地图描述对象中
        NorthAmericaMapWS.EnvelopeN pEnvelope = pMapDescription.MapArea.Extent
                 as NorthAmericaMapWS.EnvelopeN;

        double eWidth = Math.Abs(pEnvelope.XMax - pEnvelope.XMin);
        double eHeight = Math.Abs(pEnvelope.YMax - pEnvelope.YMin);
        double xFactor = (eWidth - (eWidth * 0.75)) / 2;
        double yFactor = (eHeight - (eHeight * 0.75)) / 2;
        pEnvelope.XMax = pEnvelope.XMax - xFactor;
        pEnvelope.XMin = pEnvelope.XMin + xFactor;
        pEnvelope.YMax = pEnvelope.YMax - yFactor;
        pEnvelope.YMin = pEnvelope.YMin + yFactor;

        NorthAmericaMapWS.MapExtent pMapExtext =
             new NorthAmericaMapWS.MapExtent();
        pMapExtext.Extent = pEnvelope;
        pMapDescription.MapArea = pMapExtext;

        // 保存地图描述并绘制地图
        m_sMapDesc = pMapDescription;
        DrawMap(ref pMapDescription);
```

```
        this.Cursor = Cursors.Default;
    }
    catch (Exception exception)
    {
        this.Cursor = Cursors.Default;
        MessageBox.Show(exception.Message, "An error has occurred");
    }
```

对于“缩小”工具，代码基本与放大工具一致，只是这次需要扩大范围。该工具的代码如下：

```
    this.Cursor = Cursors.WaitCursor;
    try {
        NorthAmericaMapWS.NorthAmericaMap MapServer map =
            new MapServerBrowser.NorthAmericaMapWS.NorthAmericaMap MapServer();
        map.Url = m sSelectedMap;

        NorthAmericaMapWS.MapDescription pMapDescription = m sMapDesc;

        // 得到当前范围并扩大该范围，然后将该新的范围设置到地图描述对象中
        NorthAmericaMapWS.EnvelopeN pEnvelope = pMapDescription.MapArea.Extent
                      as NorthAmericaMapWS.EnvelopeN;

        double eWidth = Math.Abs(pEnvelope.XMax - pEnvelope.XMin);
        double eHeight = Math.Abs(pEnvelope.YMax - pEnvelope.YMin);
        double xFactor = ((eWidth * 1.25) - eWidth) / 2;
        double yFactor = ((eHeight * 1.25) - eHeight) / 2;
        pEnvelope.XMax = pEnvelope.XMax + xFactor;
        pEnvelope.XMin = pEnvelope.XMin - xFactor;
        pEnvelope.YMax = pEnvelope.YMax + yFactor;
        pEnvelope.YMin = pEnvelope.YMin - yFactor;

        NorthAmericaMapWS.MapExtent pMapExtext =
               new NorthAmericaMapWS.MapExtent();
        pMapExtext.Extent = pEnvelope;
        pMapDescription.MapArea = pMapExtext;

        // 保存地图描述并绘制地图
        m sMapDesc = pMapDescription;
        DrawMap(ref pMapDescription);

        this.Cursor = Cursors.Default;
    }
    catch (Exception exception) {
        this.Cursor = Cursors.Default;
        MessageBox.Show(exception.Message, "An error has occurred");
    }
```

对于“全图”工具，需要做的是将地图的整个范围设置为当前范围。该工具的代码如下：

```
    this.Cursor = Cursors.WaitCursor;
    try {
        NorthAmericaMapWS.NorthAmericaMap_MapServer map =
```

```
            new MapServerBrowser.NorthAmericaMapWS.NorthAmericaMap_MapServer();
        map.Url = m_sSelectedMap;
        NorthAmericaMapWS.MapServerInfo mapi = map.GetServerInfo(m_sDataFrame);

        NorthAmericaMapWS.MapDescription pMapDescription = m_sMapDesc;

        // 得到地图的整个范围，并设置为当前范围
        NorthAmericaMapWS.Envelope pEnvelope = mapi.FullExtent;

        NorthAmericaMapWS.MapExtent pMapExtext =
                new NorthAmericaMapWS.MapExtent();
        pMapExtext.Extent = pEnvelope;
        pMapDescription.MapArea = pMapExtext;

        // save the map description and draw the map
        m_sMapDesc = pMapDescription;
        DrawMap(ref pMapDescription);

        this.Cursor = Cursors.Default;
    }
    catch (Exception exception)
    {
        this.Cursor = Cursors.Default;
        MessageBox.Show(exception.Message, "An error has occurred");
    }
```

“漫游”工具相对要复杂一些，它要处理鼠标的一些操作。首先在该工具中加入如下代码：

```
    ispanning = false;
    ToolStripButton btn = (ToolStripButton)e.ClickedItem;
    if (!btn.Checked) {
        ispanning = true;
        pictureBox1.Cursor = Cursors.Hand;
    }
    else {
        pictureBox1.Cursor = Cursors.Default;
    }
```

下面要处理的是图片框的鼠标按下事件的响应。首先创建该事件的响应方法，在其中加入如下代码：

```
    if (e.Button != MouseButtons.Left)
        return;

    // 判断是否选择了漫游工具
    IEnumerator benum = toolStrip1.Items.GetEnumerator();
    ToolStripItem btn = null;
    while (benum.MoveNext()) {
        btn = (ToolStripItem)benum.Current;
        if (toolStrip1.Items.IndexOf(btn) == 3 && ispanning) {
            NorthAmericaMapWS.NorthAmericaMap_MapServer map = new
```

```
            MapServerBrowser.NorthAmericaMapWS.NorthAmericaMap_MapServer();
        map.Url = m_sSelectedMap;

        NorthAmericaMapWS.MapDescription pMapDescription = m sMapDesc;
        int[] Xs = { e.X };
        int[] Ys = { e.Y };
        NorthAmericaMapWS.MultipointN mpnt = map.ToMapPoints(m sMapDesc,
               idisp, Xs, Ys) as NorthAmericaMapWS.MultipointN;
        NorthAmericaMapWS.Point[] pnta = mpnt.PointArray;
        NorthAmericaMapWS.PointN pnt = pnta[0] as NorthAmericaMapWS.PointN;
        startX = pnt.X;
        startY = pnt.Y;
        startDragX = e.X;
        startDragY = e.Y;
    }
}
```

下面要完成的是图片框的鼠标移动事件响应方法，该方法的代码如下：

```
if (e.Button != MouseButtons.Left)
    return;

IEnumerator benum = toolStrip1.Items.GetEnumerator();
ToolStripItem btn = null;
while (benum.MoveNext()) {
    btn = (ToolStripItem)benum.Current;
    if (toolStrip1.Items.IndexOf(btn) == 3 && ispanning) {
        // 拖动图像
        pictureBox1.Image = null;
        deltaDragX = startDragX - e.X;
        deltaDragY = startDragY - e.Y;
        pictureBox1.Invalidate();
    }
}
```

在上述代码中，图片框控件的 Invalidate 方法将触发该控件的 Pain 事件。该该事件响应方法中需要实现拖动图像的功能。在该事件响应方法中加入如下方法：

```
// 得到图像
Image newImage = pImage;

if (newImage != null) {
    // 创建显示图像的矩形
    Point loc = pictureBox1.Location;
    Rectangle destRect = new Rectangle(pictureBox1.Left - loc.X - deltaDragX,
                                pictureBox1.Top - loc.Y - deltaDragY,
                                pictureBox1.Width,pictureBox1.Height);
    // 在屏幕上绘制图像
    e.Graphics.DrawImage(newImage, destRect);
}
```

最后要实现的是图片框的鼠标释放事件响应。在该事件响应方法中加入如下方法：

```
if (e.Button != MouseButtons.Left)
    return;

// is pan enabled?
IEnumerator benum = toolStrip1.Items.GetEnumerator();
ToolStripItem btn = null;
while (benum.MoveNext()) {
    btn = (ToolStripItem)benum.Current;
    if (toolStrip1.Items.IndexOf(btn) == 3 && ispanning) {
        this.Cursor = Cursors.WaitCursor;
        NorthAmericaMapWS.NorthAmericaMap MapServer map = new
            MapServerBrowser.NorthAmericaMapWS.NorthAmericaMap MapServer();
        map.Url = m sSelectedMap;

        NorthAmericaMapWS.MapDescription pMapDescription = m sMapDesc;
        int[] Xs = { e.X };
        int[] Ys = { e.Y };
        NorthAmericaMapWS.MultipointN mpnt = map.ToMapPoints(m_sMapDesc,
            idisp, Xs, Ys) as NorthAmericaMapWS.MultipointN;
        NorthAmericaMapWS.Point[] pnta = mpnt.PointArray;
        NorthAmericaMapWS.PointN pnt = pnta[0] as NorthAmericaMapWS.PointN;

        double deltaX = pnt.X - startX;
        double deltaY = pnt.Y - startY;

        // 改变范围
        NorthAmericaMapWS.EnvelopeN pEnvelope =
           pMapDescription.MapArea.Extent as NorthAmericaMapWS.EnvelopeN;

        pEnvelope.XMax = pEnvelope.XMax - deltaX;
        pEnvelope.XMin = pEnvelope.XMin - deltaX;
        pEnvelope.YMax = pEnvelope.YMax - deltaY;
        pEnvelope.YMin = pEnvelope.YMin - deltaY;

        NorthAmericaMapWS.MapExtent pMapExtext =
           new NorthAmericaMapWS.MapExtent();
        pMapExtext.Extent = pEnvelope;
        pMapDescription.MapArea = pMapExtext;

        // 保存地图描述并绘制地图
        m_sMapDesc = pMapDescription;
        DrawMap(ref pMapDescription);

        deltaDragX = 0;
        deltaDragY = 0;
        pictureBox1.Invalidate();
        this.Cursor = Cursors.Default;
    }
}
```

编译并运行程序。在 Web 服务目录 URL 中输入 http://localhost/arcgis/services，然后选择“获

取 Web 服务”按钮，将在 Web 服务下拉列表框中显示 GIS 服务器中所有的地图 Web 服务。选择其中一个地图服务，便可在图片框中显示该地图服务的地图。通过“放大”、“缩小”、“漫游”按钮浏览地图。程序运行效果如图 10.9 所示。

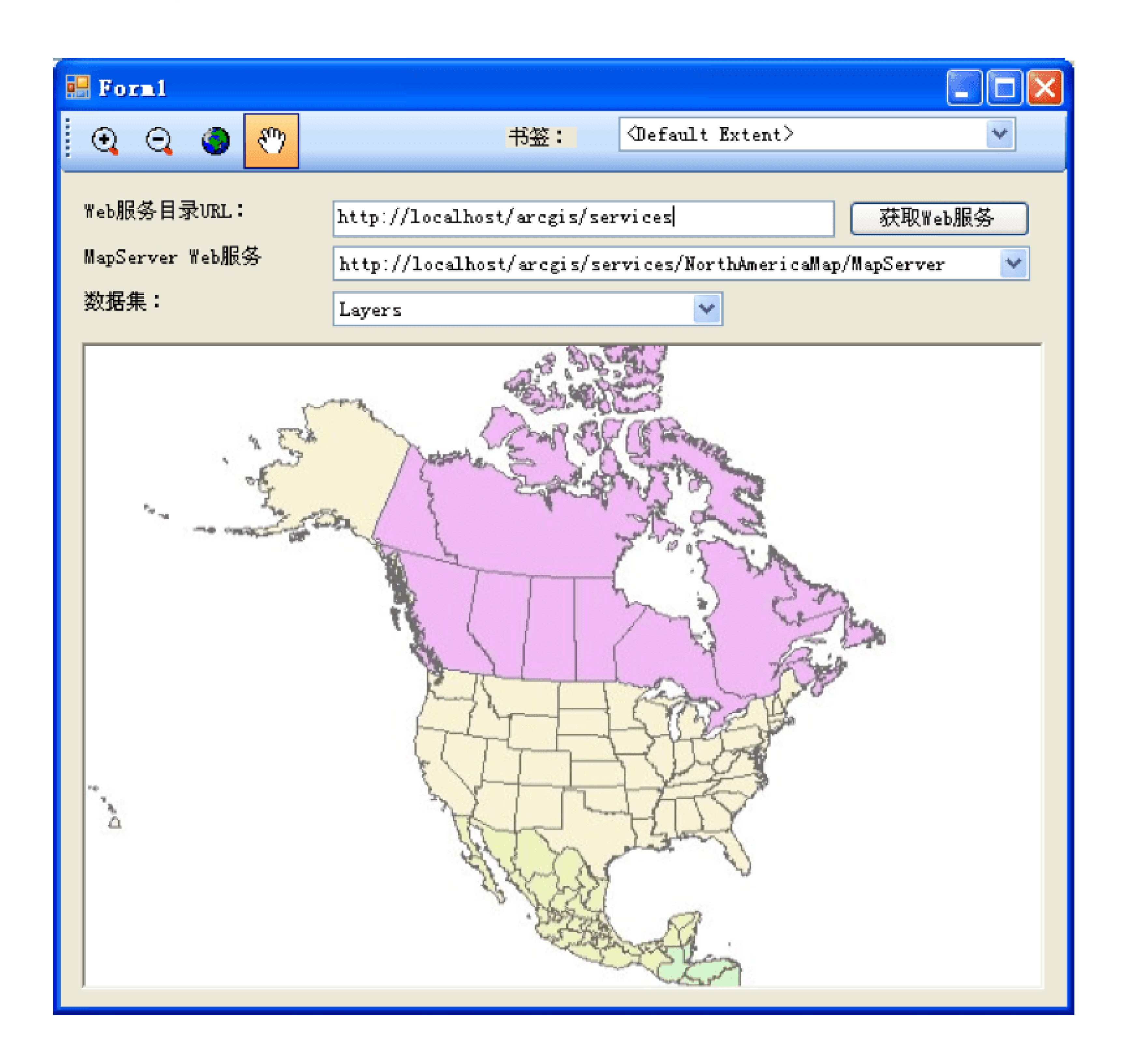


图 10.9　程序运行效果

10.2　应用性 Web 服务的创建与使用

今天，越来越多的 GIS 系统既需要访问其他系统提供的 Web 服务，也需要向其他系统提供空间信息 Web 服务。我们可以通过将 ArcGIS Server 的地图服务再次封装为 Web 服务，对其他提供更具有针对性的、调用更简单的方法。

应用性 Web 服务就是一个 ASP.NET 的 Web 服务，它的创建方式完全同于 ASP.NET 的 Web 服务的开发，只是其中的代码需要调用 ArcGIS Server 的服务。

下面我们通过一个简单的实例来演示如何创建与使用应用性 Web 服务。

10.2.1　应用性 Web 服务的创建

新建一个名为 AmericaWebService 的 ASP.NET Web 服务类型的站点。首先加入 ESRI.ArcGIS.ADF、ESRI.ArcGIS.ADF.Connection、ESRI.ArcGIS.ADF.Connection.AGS、

ESRI.ArcGIS.ADF.Web.DataSources.ArcGISServer、ESRI.ArcGIS.Geodatabase、ESRI.ArcGIS.Geometry、ESRI.ArcGIS.Server 与 ESRI.ArcGIS.System 程序集的引用。

在解决方案管理器中将 Service.asmx 更名为 QueryService。并在该类的头部加入如下命名空间的引用：

```
using ESRI.ArcGIS.esriSystem;
using ESRI.ArcGIS.Server;
using ESRI.ArcGIS.Geometry;
using ESRI.ArcGIS.ADF.Connection.AGS;
using ESRI.ArcGIS.Geodatabase;
using ESRI.ArcGIS.ADF.Web.DataSources.ArcGISServer;
using ESRI.ArcGIS.ADF;
using ESRI.ArcGIS.Carto;
```

由于我们需要使用服务器上下文，因此首先加入得到该服务器上下文的辅助方法。代码如下：

```
private IServerContext getServerContext()
{
    string servername = "localhost";
    string mapserverobject = "USAMap";
    IServerObjectManager serverManager;

    // *** 使用 Web ADF 公有 API
    Identity identity = new Identity("username", "password", null);
    AGSServerConnection gisconnection = new AGSServerConnection(servername,
        identity);
    gisconnection.Connect();

    serverManager = gisconnection.ServerObjectManager;


    IServerContext mapContext =
        serverManager.CreateServerContext(mapserverobject, "MapServer");
    return mapContext;
}
```

在上面的代码中，我们通过 AGSServerConnection 连接对象，建立与 GIS 服务器的连接，然后通过 CreateServerContext 方法得到 USAMap 地图服务的上下文。

然后加入一个名为 PointQuery 的 Web 方法的代码框架。该方法用于实现从 USAMap 地图服务的第二个图层（即美国州行政图层）中进行点查询，并返回查询到的州的名称：

```
[WebMethod]
public string PointQuery(double x, double y)
{
}
```

在 PointQuery 方法中，加入如下代码：

```
IServerContext serverContext = getServerContext();
IMapServer mapServer = serverContext.ServerObject as IMapServer;
```

```
IMapServerInfo mapInfo = mapServer.GetServerInfo(mapServer.DefaultMapName);
IMapDescription mapDesc = mapInfo.DefaultMapDescription;

ISpatialFilter spatialFilter =
    serverContext.CreateObject("esriGeodatabase.SpatialFilter")
    as ISpatialFilter;
IPoint geometry = serverContext.CreateObject("esriGeometry.Point") as IPoint;
geometry.X = x;
geometry.Y = y;
spatialFilter.Geometry = geometry;
spatialFilter.SpatialRel = esriSpatialRelEnum.esriSpatialRelIntersects;
IRecordSet rs = mapServer.QueryFeatureData(
                    mapDesc.Name, 1, (IQueryFilter)spatialFilter);
if (rs == null)
    return "";
ICursor cursor = rs.get_Cursor(true);
IRow row = cursor.NextRow() as IRow;
if (row == null)
    return "";

int fieldIndex = rs.Table.FindField("STATE_NAME");
string stateName = row.get_Value(fieldIndex).ToString();

// 释放服务器上下文
serverContext.ReleaseContext();

return stateName;
```

在 PointQuery 方法中，首先调用 getServerContext 方法得到服务器上下文，从该上下文对象的 ServerObject 属性中得到地图服务器对象。然后利用服务器上下文的 CreateObject 方法创建空间查询对象。接着调用地图服务器对象的 QueryFeatureData 方法实现空间查询。最后返回查询结果中字段为 STATE_NAME 的结果。

10.2.2　应用性 Web 服务的应用

由于应用性 Web 服务是一标准的 ASP.NET Web 服务，因此该服务的应用很简单。

新增加一名为 WebServiceTest 的控制台。加入 10.2.1 节创建的 Web 服务的 Web 引用。

在 Program 类的 Main 方法中加入如下代码，用于调用 Web 服务的 PointQuery 方法，实现查询：

```
QueryService.QueryService service = new QueryService.QueryService();
string stateName = "";
try
{
    stateName = service.PointQuery(-111.6125, 35.912);
    Console.WriteLine("查询成功，查询结果为：");
    Console.WriteLine(stateName);
```

```
    }
    catch
    {
        Console.WriteLine("查询失败");
    }

    Console.ReadLine();
```

编译并运行该程序。程序运行结果如图 10.10 所示。



图 10.10　应用 Web 服务的方法执行查询

从 Main 方法中的代码可以看出，由于我们将空间查询的服务进行了进一步的封装，因此在客户端的调用非常简单，只需要新创建一个对象，然后调用方法即可实现空间查询。

第 11 章



安全、部署与性能调优

对于 Web 应用程序，功能的开发只占整个工作量的一部分，还需要考虑的是应用程序的安全与性能。特别是对于 Web GIS，由于通常系统中的空间数据安全要求比较高，而且由于空间数据的数据量比较大，因此特别需要开发人员在应用程序的安全与性能上花费较大精力。

因此，本章就来介绍 Web 应用程序的安全、部署以及性能调优事宜。

11.1 应用程序的安全

对于 Web 开发人员来说，保证网站的安全是一个关键而又复杂的问题。保护某个站点需要进行仔细的规划，网站管理员和程序员必须清楚地了解有关保证他们站点安全的选项。

ASP.NET 与 Microsoft .NET Framework 及 Microsoft Internet 信息服务（IIS）协同工作，以提供 Web 应用程序安全性。若要帮助保护 ASP.NET 应用程序，应该执行下面两个基本功能：

- 身份验证。验证用户不是假冒的。应用程序获取用户的凭据（各种形式的标识，如用户名和密码）并通过某些授权机构验证那些凭据。如果这些凭据有效，则将提交这些凭据的实体视为通过身份验证;
- 授权。通过对已验证身份授予或拒绝特定权限来限制访问权限。

11.1.1 使用成员资格管理用户

ASP.NET 成员资格（Membership 类）提供了一种验证和存储用户凭据的内置方法。因此，ASP.NET 成员资格可管理网站中的用户身份验证。可以将 ASP.NET 成员资格与 ASP.NET Forms 身份验证或 ASP.NET 登录控件一起使用以创建一个完整的用户身份验证系统。

ASP.NET 成员资格支持下列功能：

（1）创建新用户和密码。

（2）将成员资格信息（用户名、密码和支持数据）存储在 Microsoft SQL Server、Active Directory 或其他数据存储区。

（3）对访问站点的用户进行身份验证。可以以编程方式验证用户，也可以使用 ASP.NET 登录控件创建一个只需很少代码或无需代码的完整身份验证系统。

（4）管理密码，包括创建、更改和重置密码。根据选择的成员资格选项不同，成员资格系统还可以提供一个使用用户提供的问题和答案的自动密码重置系统。

（5）公开经过身份验证的用户的唯一标识，可以在应用程序中使用该标识，也可以将该标识与 ASP.NET 个性化设置和角色管理（授权）系统集成。

（6）指定自定义成员资格提供程序，可以使用自己的代码管理成员资格及在自定义数据存储区中维护成员资格数据。

若要使用成员资格，必须首先为站点配置成员资格。主要分为下面的步骤：

（1）将成员资格选项指定为网站配置的一部分。默认情况下，成员资格处于启用状态。还可以指定要使用哪个成员资格提供程序。（实际上，这意味着指定要存储成员资格信息的数据库的类型。）默认提供程序使用 Microsoft SQL Server 数据库。还可以选择使用 Active Directory 存储成员资格信息，或者可以指定自定义提供程序。

（2）将应用程序配置为使用 Forms 身份验证。通常指定应用程序中的某些页或文件夹受到保护，并只能由经过身份验证的用户访问。

（3）为成员资格定义用户账户。可以通过多种方式执行此操作。可以创建一个“新用户”ASP.NET 网页，在该网页中收集用户名和密码（及电子邮件地址（可选）），然后使用一个名为 CreateUser 的成员资格函数在成员资格系统中创建一个新用户。或者，另一个更常用的，特别是开发期间更常用的，是使用网站管理工具，该工具提供了一个用于创建新用户的类似向导的界面。

其实不仅可以用网站管理工具创建用户，还可用该工具配置成员资格与身份验证。可以通过 Website 菜单的 ASP.NET Configuration 命令，打开该工具。首先进入 Home 选项卡，在其中选择 Security 选项卡（此时该工具自动在网站的 App_Data 目录下创建 Microsoft SQL Server 速成版数据库文件），进入如图 11.1 所示的界面。在该界面中可选通过“Select authentication type”配置验证类型。

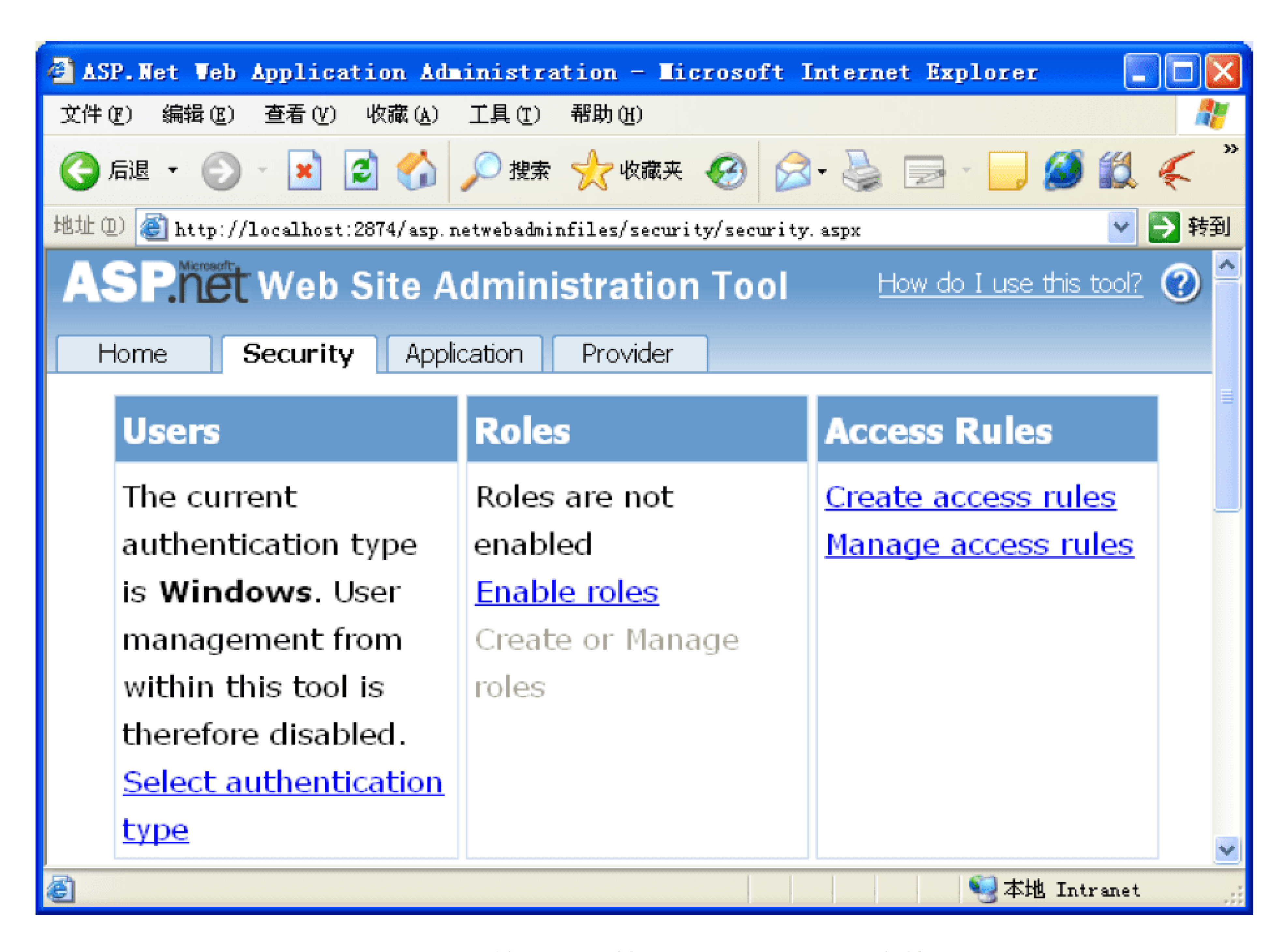


图 11.1　利用网站管理工具配置成员资格

现在，就可以使用成员资格对应用程序中的用户进行身份验证。大多数情况下，将需要提供一个登录窗体，它可能是一个单独页或主页上的一个专用区域。可以使用 ASP.NET TextBox 控件手动创建登录窗体，也可以使用 ASP.NET 登录控件。由于已将应用程序配置为使用 Forms 身份验证，因此在未经验证的用户请求一个受保护的页面时，ASP.NET 将自动显示登录页。

11.1.2　使用角色管理授权

建立角色的主要目的是为应用系统提供一种管理用户组访问规则的便捷方法。创建用户，然后将用户分配到角色。典型的应用是创建一组要限制为只有某些用户可以访问的页面。通常的做法是将这些受限制的页面单独放在一个文件夹内。然后，可以使用网站管理工具定义允许和拒绝访问受限文件夹的规则。例如，可以配置站点以使成员和经理可以访问受限文件夹中的页面，并拒绝其他

所有用户的访问。如果未被授权的用户尝试查看受限制的页面，该用户会看到错误消息或被重定向到指定的页面。

若要使用 ASP.NET 角色管理，则需要使用如下所示的设置在应用程序的 Web.config 文件中启用它：

```
<roleManager
    enabled="true"
    cacheRolesInCookie="true" >
</roleManager>
```

角色的典型应用是建立规则，用于允许或拒绝对页面或文件夹的访问。可以在 Web.config 文件的 authorization 元素部分中设置此类访问规则。下面的示例允许 members 角色的用户查看名为 memberPages 的文件夹中的页面，同时拒绝任何其他用户的访问：

```
<configuration>
   <location path="memberPages">
      <system.web>
          <authorization>
             <allow roles="members" />
             <deny users="*" />
          </authorization>
        </system.web>
      </location>
   <!-- 其他配置信息的设置-->
<configuration>
```

但是 ASP.NET 中的成员资源与角色的配合使用只能到页面层次，对于 GIS 一些特有的功能，例如对某些用户需要限制查询操作，或限制某些用户浏览某些图层的操作，ASP.NET 没有提供，也不可能提供。因此对于这些 GIS 特殊的功能，需要程序员编码实现。

下面的代码演示了如果根据用户的角色设置图层的可见性、隐藏工具与任务。

```
protected void Page PreRenderComplete(object sender, EventArgs e) {
    // 检查登录用户的角色
    if (!User.IsInRole(fullyEnabledRole)) {
        // 1) 从地图以及 Toc 中删除某图层
        HideLayer(Map1, layerToHide);

        // 2) 隐藏地图信息工具
        RemoveItemFromToolbar(Toolbar1, toolbarItemToRemove);

        // 3) 隐藏任务 1
        RemoveTask(TaskManager1, taskToRemove);
    }
}
```

将某图层设置为不可见的代码如下：

```
private void HideLayer(Map map, string layerName)
{
    // 遍历所有 Map Functionalities
```

```
    foreach (IMapFunctionality mapFunct in map.GetFunctionalities()) {
        string layerId = GetLayerId(layerName, mapFunct);

        // 设置图层不可见
        if (!String.IsNullOrEmpty(layerId))
            mapFunct.SetLayerVisibility(layerId, false);
    }

    // 找到和 Map 关联的 Toc
    Toc tocCtrl = FindControlOfType(typeof(Toc), Page.Controls) as Toc;
    if (tocCtrl != null && tocCtrl.BuddyControl == map.ID) {
        // 移除之前隐藏图层在 Toc 中的节点
        foreach (TreeViewPlusNode resource in tocCtrl.Nodes) {
            TreeViewPlusNode nodeToRemove = null;
            foreach (TreeViewPlusNode layerNode in resource.Nodes) {
                TreeViewPlusNode matchNode =
                   FindNodeRecursive(layerNode, layerName);
                if (matchNode != null) {
                    nodeToRemove = matchNode;
                    break;
                }
            }
            if (nodeToRemove != null)
                nodeToRemove.Remove();
        }
    }
}
```

上述方法利用了 GetLayerId 方法，根据图层名称获取图层的 id。该方法的代码如下：

```
private string GetLayerId(string layerName,
IMapFunctionality mapFunctionality)
{
    string layerId = String.Empty;
    string[] layerIDs, layerNames;

    // 查询图层名称
    mapFunctionality.GetLayers(out layerIDs, out layerNames);
    for (int i = 0; i < layerIDs.Length; i++) {
        if (layerNames[i] == layerName) {
            layerId = layerIDs[i];
            break;
        }
    }
    return layerId;
}
```

在 HideLayer 方法中还用到了 FindNodeRecursive 方法，用于查找指定节点。该方法的代码如下：

```
private TreeViewPlusNode FindNodeRecursive(TreeViewPlusNode node,
```

```
    string nodeName) {
        if (node.Text == nodeName) {
            return node;
        }
        foreach (TreeViewPlusNode node2 in node.Nodes) {
            TreeViewPlusNode childNode = FindNodeRecursive(node2, nodeName);
            if (childNode != null) {
                return childNode;
            }
        }
        return null;
    }
```

从工具栏中删除某工具的代码如下：

```
    private void RemoveItemFromToolbar(Toolbar toolbar, string toolbarItemName)
    {
        for (int i = 0; i < toolbar.ToolbarItems.Count; i++) {
            if (toolbar.ToolbarItems[i].Name == toolbarItemName) {
                toolbar.ToolbarItems.RemoveAt(i);
                break;
            }
        }
    }
```

删除指定名称任务的代码如下：

```
    private void RemoveTask(TaskManager taskManager, string taskName) {
        string menuId = "";
        foreach (Control ctl in taskManager.Controls) {
            if (ctl.ID == taskToRemove) {
                // 得到任务标题，以便从菜单中删除
                FloatingPanel queryTask = ctl as FloatingPanel;
                if (queryTask != null)
                    menuId = queryTask.Title;
                // 从任务管理器中删除该任务
                taskManager.Controls.Remove(ctl);
            }
        }
        // 同时需要从菜单中删除该任务对应的菜单
        // 找到任务管理器连接的菜单控件
        Control mCtl =
            FindControlRecursive(taskManager.Page, taskManager.BuddyControl);
        if (mCtl != null && mCtl is Menu) {
            Menu menu = (Menu)mCtl;
            for (int i = 0; i < menu.Items.Count; i++) {
                if (menu.Items[i].Text == menuId) {
                    menu.Items.RemoveAt(i);
                    break;
                }
            }
        }
```

```
    }
```

上述代码利用了 FindControlRecursive 方法查找指定控件，该方法的代码如下：

```
public static Control FindControlOfType(Type type, ControlCollection controls)
{
    foreach (Control ctl in controls) {
        if (type.IsInstanceOfType(ctl)) {
            return ctl;
        }
        else if (ctl.HasControls()) {
            Control childCtl = FindControlOfType(type, ctl.Controls);
            if (childCtl != null) {
                return childCtl;
            }
        }
    }
    return null;
}
```

11.1.3 使用受保护的配置加密配置信息

不用可读或容易解码的格式存储高度敏感的信息，这也是保护应用程序所要注意的一部分。敏感信息的示例包括用户名、密码、连接字符串和加密密钥。将敏感信息以不可读的格式存储，可以使攻击者很难获得对敏感信息的访问权限（即使攻击者获得了对文件、数据库或其他存储位置的访问权限），从而可以增强应用程序的安全性。

ASP.NET 应用程序中存储敏感信息的主要位置之一是 Web.config 文件。为了帮助保护配置文件中的信息，ASP.NET 提供了一项称为“受保护配置”的功能，可用于加密配置文件中的敏感信息。

可以使用 aspnet_regiis.exe 对 Web.config 文件的节进行加密并管理加密密钥。ASP.NET 在处理文件时对配置文件进行解密。因此，解密不需要任何附加代码。

11.1.4 显示安全的错误信息

在应用程序显示错误信息时，不应该泄露有助于恶意攻击用户系统的信息。例如，如果应用程序试图登录数据库时没有成功，则显示的错误信息不应该包括它正在使用的用户名。

有许多方法可以控制错误信息，包括：

（1）将应用程序配置为不向远程用户显示详细错误信息（远程用户是指在 Web 服务器计算机上工作但未请求页的用户），也可以选择将错误重定向到应用程序页。

（2）只要可行就包括错误处理，并编写自定义的错误信息。在错误处理程序中，可以进行测试以确定用户是否为本地用户并做出相应的响应。

（3）在捕捉所有未处理异常并将它们发送到一般错误页的页级别或应用程序级别上，创建全

局错误处理程序。这样，即使没有预料到某个问题，至少用户不会看到异常页。

1. 将应用程序配置为不向远程用户显示错误

在应用程序的 Web.config 文件中，对 customErrors 元素进行以下更改：

- 将 mode 属性设置为 RemoteOnly。这就将应用程序配置为仅向本地用户（即开发人员）显示详细的错误。
- 包括指向应用程序错误页的 defaultRedirect 属性。
- 包括将错误重定向到特定页的 <error> 元素。例如，可以将标准 404 错误（未找到页）重定向到自定义的应用程序页。

下面的代码示例显示 Web.config 文件中的典型 customErrors 块。

```
<customErrors mode="RemoteOnly" defaultRedirect="AppErrors.aspx">
   <error statusCode="404" redirect="NoSuchPage.aspx"/>
   <error statusCode="403" redirect="NoAccessAllowed.aspx"/>
</customErrors>
```

2. 包含错误处理

在任何可能生成错误的语句周围使用一个 try-catch 块。

另一方面，可以选择使用 IsLocal 属性对本地用户进行测试并相应地修改错误处理。值 127.0.0.1 等效于 localhost，指示浏览器与 Web 服务器位于同一台计算机上。

下面的代码示例显示一个错误处理块。如果发生错误，则用有关消息的详细信息加载 Session 状态变量，然后应用程序显示可以读取 Session 变量并显示错误的页。（有意写入此错误以便不向用户提供任何可利用的详细信息。）如果用户是本地用户，则提供不同的错误详细信息。在 finally 块中，释放开放式资源。

```
try {
    sqlConnection1.Open();
    sqlDataAdapter1.Fill(dsCustomers1);
}
catch (Exception ex) {
    if(Request.IsLocal) {
        Session["CurrentError"] = ex.Message; }
    else {
        Session["CurrentError"] = "Error processing page.";
    }
    Server.Transfer("ApplicationError.aspx");
}
finally {
    this.sqlConnection1.Close();
}
```

3. 创建全局错误处理程序

若要在页面中创建全局处理程序，则需要创建 System.Web.UI.TemplateControl.Error 事件的处

理程序。若要创建应用程序范围的错误处理程序，这需要在 Global.asax 文件中将代码添加到 System.Web.HttpApplication.Error 事件。只要页面或应用程序中发生未处理的异常，就会相应地调用这些方法。可以从 GetLastError 方法获取有关最新错误的信息。

下面的代码示例显示一个处理程序，它获取有关当前错误的信息，将其放在 Session 变量中，然后调用可以提取和显示错误信息的一般性错误处理页。

```
protected void Application_Error(Object sender, EventArgs e) {
   Session["CurrentError"] = "Global: " + Server.GetLastError().Message;
   Server.Transfer("lasterr.aspx");
}
```

11.1.5　防止拒绝服务威胁

恶意用户危害应用程序的一种间接方式是使其不可用。恶意用户可以使应用程序太忙而无法为其他用户提供服务，或者仅仅使应用程序出现故障。请遵循下面这些原则：

（1）关闭或释放使用的任何资源。例如，在使用完毕后，始终关闭数据连接和数据读取器，而且始终关闭文件。

（2）使用错误处理机制（例如，try-catch 块）。包含 finally 块，以便万一失败就可以在其中释放资源。

（3）将 IIS 配置为使用调节，这样可以防止应用程序消耗过多的 CPU。

（4）在使用或存储用户输入之前，测试它的大小限制。

（5）对数据库查询设置大小保护措施，以防止大型查询耗尽系统资源。

（6）如果文件上载是应用程序的一部分，则对它们的大小加以限制。可以使用类似下面代码示例的语法在 Web.config 文件中设置限制（其中 maxRequestLength 值以千字节为单位）：

```
<configuration>
   <system.web>
      <httpRuntime maxRequestLength="4096" />
   </system.web>
</configuration>
```

还可以使用 RequestLengthDiskThreshold 属性来减少大型上载和窗体发布所需的内存开销。

11.2　应用程序的部署

由于使用 Web ADF 的 Web 应用程序是一个典型的 ASP.NET 程序，因此部署比较简单，最简单的方法是将开发环境中的程序复制到目标计算机的 IIS 目录中即可。但是我们可以使用 ASP.NET 2.0 的预编译功能提高系统响应速度以及保护源代码。

默认情况下，在用户首次请求资源（如网站的一个页面）时，将动态编译 ASP.NET 网页和代码文件。第一次编译页和代码文件之后，会缓存编译后的资源，这样将大大提高随后对同一页提出的请求的效率。

ASP.NET 还可以预编译整个站点，然后再提供给用户使用。这样做有很多好处，其中包括：

（1）可以加快用户的响应时间，因为页和代码文件在第一次被请求时无须编译。这对于经常更新的大型站点尤其有用。

（2）可以在用户看到站点之前识别编译时的错误。

（3）可以创建站点的已编译版本，并将该版本部署到成品服务器，而无须使用源代码。

ASP.NET 提供了两个预编译站点选项：

（1）预编译现有站点。如果希望提高现有站点的性能并对站点执行错误检查，那么此选项十分有用。

（2）针对部署预编译站点。此选项将创建一个特殊的输出，可以将该输出部署到成品服务器。

另外，可以预编译一个站点，使它成为只读的或可以更新的站点。

当仅针对部署进行预编译时，编译器实质上将基于正常情况下在运行时编译的所有 ASP.NET 源文件来生成程序集。其中包括页中的程序代码、.cs 和 .vb 类文件以及其他代码文件和资源文件。编译器将从输出中移除所有源代码和标记。在生成的布局中，为每个.aspx 文件生成编译后的文件（扩展名为.compiled），该文件包含指向该页相应程序集的指针。

要更改网站（包括页的布局），必须更改原始文件，重新编译站点并重新部署布局。唯一的例外是站点配置；可以更改成品服务器上的 Web.config 文件，而无须重新编译站点。

“仅针对部署进行预编译”选项不仅为 ASPX 页面提供了最大程度的保护，还提供了最佳启动性能。利用 ASP.NET 编译工具 (Aspnet_compiler.exe) 可以就地对应用程序进行预编译。

就地预编译 ASP.NET 网站的命令如下：

```
aspnet_compiler -v /virtualPath
```

其中 virtualpath 参数指示网站的 Internet 信息服务 (IIS) 虚拟路径。

预编译 ASP.NET 网站以进行部署的命令如下：

```
aspnet_compiler -v virtualPath targetPath
```

使用-u 选项将以下面的方式编译应用程序：可以对编译好的应用程序中的某些文件做出更改，而无须重新编译该应用程序。Aspnet_compiler.exe 区分静态和动态文件类型之间的不同，并在创建生成的应用程序时，以不同的方式对它们进行处理。

- 静态文件类型是指那些没有关联的编译器或生成提供程序的文件类型，例如具有 .css、.gif、.htm、.html、.jpg、.js 等扩展名的文件。这些文件只是复制到目标位置，并且在保留的目录结构中保持它们的相对位置；
- 动态文件类型是指那些具有关联的编译器或生成提供程序的文件类型，包括具有 ASP.NET 特定文件扩展名（如 .asax、.ascx、.ashx、.aspx、.browser、.master 等）的文件。ASP.NET 编译工具从这些文件生成程序集。如果省略-u 选项，该工具还会创建具有文件扩展名.COMPILED 的文件，这些文件将原始源文件映射到它们的程序集。为确保保留应用程序源的目录结构，该工具在目标应用程序的相应位置中生成占位符文件。

必须使用-u 选项，以指示可以修改编译好的应用程序的内容。否则，将忽略后续修改或者导

致运行时错误。

11.3　性能调优

ArcGIS Server 产品的强大功能往往给了有些用户带来一定的认识误区。有些用户会认为可以把 C/S 程序的使用模式和数据照搬到 ArcGIS Server 的应用系统中。ArcGIS Server 的应用是 B/S 的应用，涉及到数据库、Web 服务器、ArcGIS 服务器与浏览器，任何一个环节都有可能出现问题，所以 ArcGIS Server 系统需要很好的设计和规划，简单的功能移植只能得到低效率的 ArcGIS Server 应用。

其实 ArcGIS Server 的应用系统往往都比较大，特别是数据量都比较大。用户都希望把大量的数据都通过网络共享给广大的浏览器用户。由于 ArcGIS Server 简单功能的系统的构建是非常简单的，用户很容易就搭建出来了，因此当用户把大量的数据往 ArcGIS Server 系统上搬时并发现速度非常慢时，用户往往会认为是 ArcGIS Server 产品的速度慢，因此性能调整的考虑角度也仅仅从 ArcGIS Server 产品角度入手，这也给 ArcGIS Server 系统的性能调整带来一定的误区，忽略了 ArcGIS Server 系统中的重要组成部分 —— 数据的性能。

当然，不同应用系统有不同的情况，因此在性能调整方面都要看具体系统情况，这里简单介绍 ArcGIS Server 应用性能调整的几个大的方面。

11.3.1　数据方面

ArcGIS Server 应用毕竟是 B/S 程序，网络传输问题很容易成为它的性能瓶颈，除了增加网络带宽之外，当然也尽量要减少数据量，主要的原则是按需使用数据。

比如在对数据进行了分析之后，发现数据节点非常密，在不影响数据的浏览精度的情况下使用简化操作，减少数据量。

如果是文件方式的话，就没有什么可调整性，如果是 SDE 中的数据，那么主要从两个方面对数据进行调整，一个是属性字段的索引，一个是空间索引大小的调整。

- ❑ 属性字段索引：特别是对于经常要进行搜索的字段都要进行索引的建立。
- ❑ 空间索引大小的调整：会非常影响数据的浏览和空间查询的速度。空间索引大小的调整要依照数据的每个单元的大小而定。一般建立 2 级就可以，第二级是第一级的 4~5 倍，第一级是要素类中大多数要素的大小。这个设定的效果可以在桌面产品中进行验证。

二维的地图服务都是通过发布 mxd 文件的，因此 mxd 的文档组织也非常重要。主要包括减少图层数量，图层的按比例显示，减少复杂符号，减少注记等。

11.3.2　服务方面

如果应用中要包含大量的图层，比如上百个，不要把所有的图层都放在一个服务中，可以把上

百个图层进行拆分，做成几个服务，然后在应用中进行组合，这样就可以有多个进程来完成，从而减少处理时间。

另外，这样拆分的另一个好处就是可以根据不同的用户使用不同的服务，没有必要每个人都添加所有的数据，这也是按需使用数据的原则。

服务设置也有两个方面，一个是服务的池化和非池化设置，一个是服务的缓存的设置。池化服务的效率比非池化服务的效率要高，而且池化服务可以创建缓存来更进一步提供高速度。

11.3.3 应用系统的配置与部署方面

把 ArcGIS Server 应用进行分布式部署，由一个 SOM 和多个 SOC 组成，SOM 起到负载均衡的作用，具体的请求由 SOC 来完成。大型系统利用多台服务器的应用系统可采用如图 11.2 所示的结构。

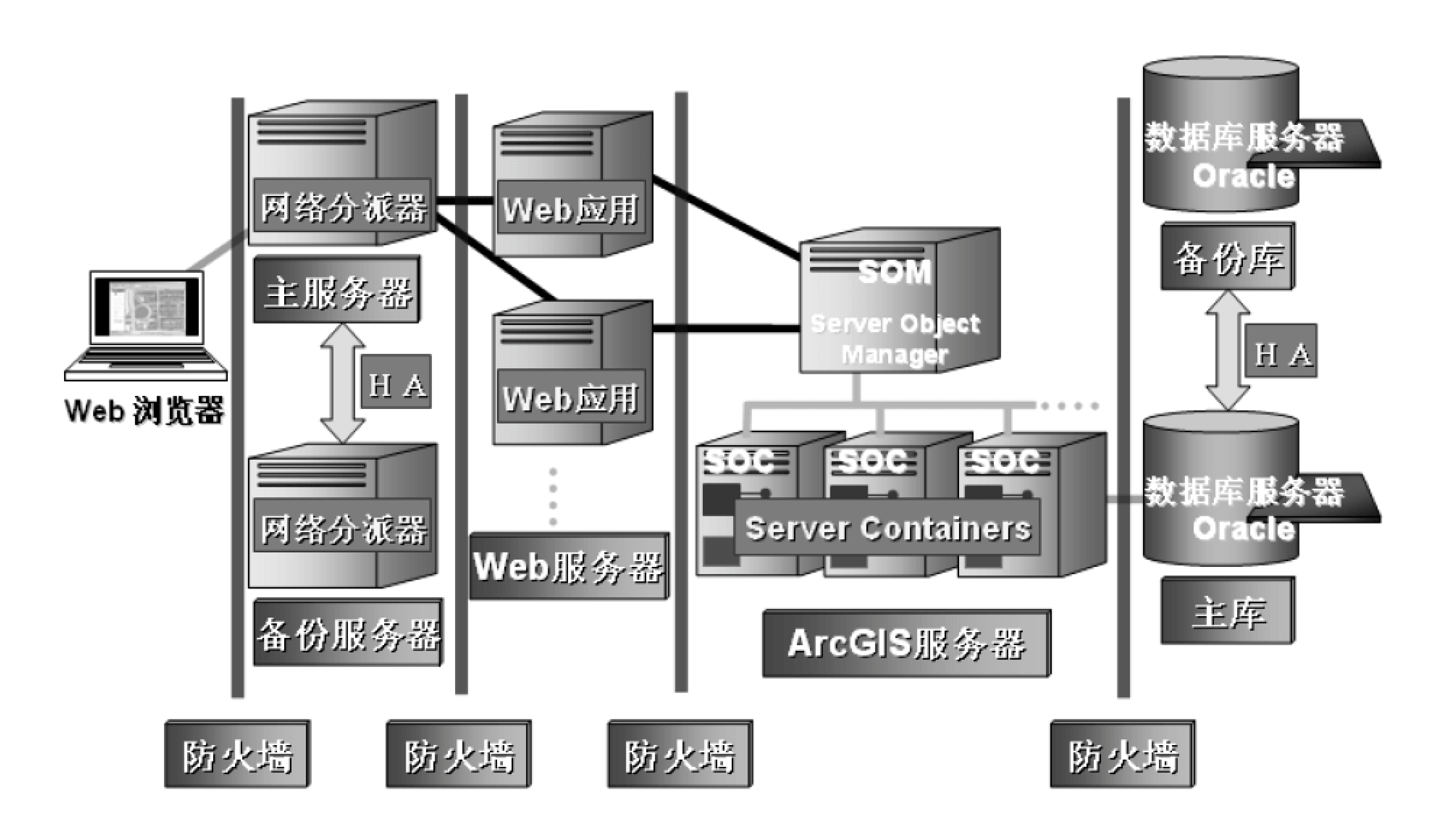


图 11.2 大型 Web GIS 应用系统部署方案

在图 11.2 中，网络分派器将让多个 Web 服务器像一个单一系统一样运行。该软件驻留于服务器和 Internet 之间，提供负载平衡和动态配置，若其中某个服务器脱机时它还提供失效转移（fail-over）功能。ArcGIS 服务器宿主各种 GIS 资源，例如地图资源、地理编码资源等，并将它们封装为服务提供给客户端应用。GIS 服务器本身包括两部分，分别是 SOM 与 SOCs，一个 SOM 和一个或多个 SOC。客户端发送请求到 SOM，SOM 将分配的资源提供给客户端，通过 SOM 对 SOC 进行调度与管理。

该部署架构具有如下特点：

- 高可靠性（HA）。利用集群管理软件，当主服务器发生故障时，备份服务器能够自动接管主服务器的工作，并及时切换过去，以实现对用户的不间断服务。
- 高性能计算（HP）。即充分利用集群中的每一台计算机的资源，实现复杂运算的并行处理。

- 负载平衡。即把负载压力根据某种算法合理分配到集群中的每一台计算机上，以减轻主服务器的压力，降低对主服务器的硬件和软件要求。

11.3.4 .NET 程序代码调优

对程序代码调优可以遵循以下几个原则。

（1）减小没有必要的操作

对象的创建是个很昂贵的工作，所以应当尽量减少对象的创建，在需要的时候声明它，初始化，不要重复初始化一个对象，尽量能做到再使用，而用完后置null有利于垃圾收集。让类实现Cloneable接口，同时采用工厂模式，将减少类的创建，每次都是通过 clone()方法来获得对象。另外使用接口也能减少类的创建。对于成员变量的初始化也应尽量避免, 特别是在一个类派生另一个类时。

异常抛出对性能不利。抛出异常首先要创建一个新的对象。只要有异常被抛出，就必须调整调用堆栈，因为在处理过程中创建了一个新的对象。异常只能用于错误处理，不应该用来控制程序流程。

此外，关闭 Debug 输出，尽量少用串行化、同步操作和耗时昂贵的服务。

（2）使用合适的类型

当原始类型不能满足我们要求时，使用复杂类型。在涉及到字符运算时，使用 StringBuffer。

带有 final 修饰符的类是不可派生的，如果指定一个类为 final，则该类所有的方法都是 final。编译器会寻找机会内联所有的 final 方法，这将能够使性能平均提高 50%。类的属性和方式使用 final 或者 static 修饰符也是有好处的。

调用方法时传递的参数以及在调用中创建的临时变量都保存在栈中，速度较快。所以尽量使用局部变量。

ArrayList 和 Vector，HashMap 和 Hashtable 是经常用到的类，前者不支持同步，后者支持同步，前者性能更好，大多数情况下选择前者。

（3）使用嵌入式资源

ASP.NET 2.0 允许开发人员将资源嵌入到.NET 程序集中。例如，我们创建了使用一些图像文件的自定义任务，当然并且希望该任务可以在多个应用系统中重复使用，这时我们可以将这些图像文件嵌入到动态链接库中，这样可以避免既需要复制动态链接库，又需要复制图像文件。同样可以将 JavaScript 与其他类型的资源以及页面嵌入到程序集中。使用时，可以通过 URL 来获取该资源。该 URL 使用 WebResource.axd 处理程序来获取嵌入资源。

（4）尽量使用缓存

使用缓存能大大提高系统的性能。我们在第 4 章中介绍了如何使用缓存。

11.3.5 ArcGIS Server 的代码的调优

可以通过设置超时或查询操作的参数等来优化 ArcGIS Server 代码。

1. 设置超时

使用 Web ADF 的应用程序可以在三个层面考虑超时：Web ADF 超时（客户端）、ASP.NET 会话超时（Web 端）、数据源超时（GIS 服务器端）。

display_common.js 文件中的如下代码设置了 Web ADF 超时：

```
var maximumLapseTime = 10;
```

该超时值的单位为分钟。要注意的是，默认时，该文件作为 Web 资源保存到了客户端。要在 Web ADF 站点中改变该值，需要在 Web ADF 控件中改变下面的属性：

```
UseDefaultWebResources = False
WebResourceLocation = <JavaScript 文件所在的相对路径>
```

将 UseDefaultWebResources 设置为 False，意味着将使用 WebResourceLocation 属性指定的值来定位 JavaScript 文件。WebResourceLocation 的默认值为/aspnet_client/ESRI/WebADF/JavaScript/。

ASP.NET 会话超时可以在 web.config 文件中设置。默认值为 20 分钟。要在应用程序中改变该值，在 web.config 文件中的<system.web>元素部分加入如下配置：

```
<sessionState timeout="10"/>
```

上述值将降低使用非池化 ArcGIS Server 服务的 Web 应用程序的效率。这是由于默认会话超时是 20 分钟，如果 Web 应用程序中，在 20 分钟里启动了多个会话，每个会话试图创建并拥有一个服务器对象实例。如果 GIS 服务器上没有更多的实例可用，就会发生许多问题。如果需要在 Web 应用程序中使用非池化 ArcGIS Server 服务，可以将会话超时值设置为更短时间，或重新启动服务器对象。

数据源提供程序管理数据源超时。例如，ArcIMS 管理员可改变处理请求的空间服务器的时间分配。

2. 查询参数记录

在进行空间查询操作时，可以通过将 MaxRecords 设置为较小的值来提高查询效率。此外，对于只需要返回属性设置的查询，通过 ReturnADFGeometries 属性设置为 false，指定查询结果不返回几何图形数据，可以大大提高服务器的响应速度。

3. 维护用户状态

Web 本身是无状态的，每个请求分别与服务器连接。但是对于 Web 应用程序，需要在请求之间提供连续性。例如，地图应用程序需要记忆在用户界面上当前显示的地图的返回，只有这样才能正确执行放大、缩小与漫游操作。

同样，可以在三个不同的层面来保存用户状态：客户端、Web 端与 GIS 服务器端。在 Web ADF 9.2 中，主要使用 Web 端来维护状态信息。只有在使用非池化 ArcGIS Server 资源时，GIS 服务器才维护状态。

有时还需要在不同的会话之间保存状态。例如，可能需要保存用户地图当前的显示范围、数据中选择的要素，以便用户再次登录时恢复用户界面。Web ADF 没有提供保存该类型的状态的功能，这需要开发人员自己完成。